SEBA Master: Web Application Engineering

Tutorial: Building Rest-Enabled Back-End Services

26th of May 2020, Ingo Glaser, Munich

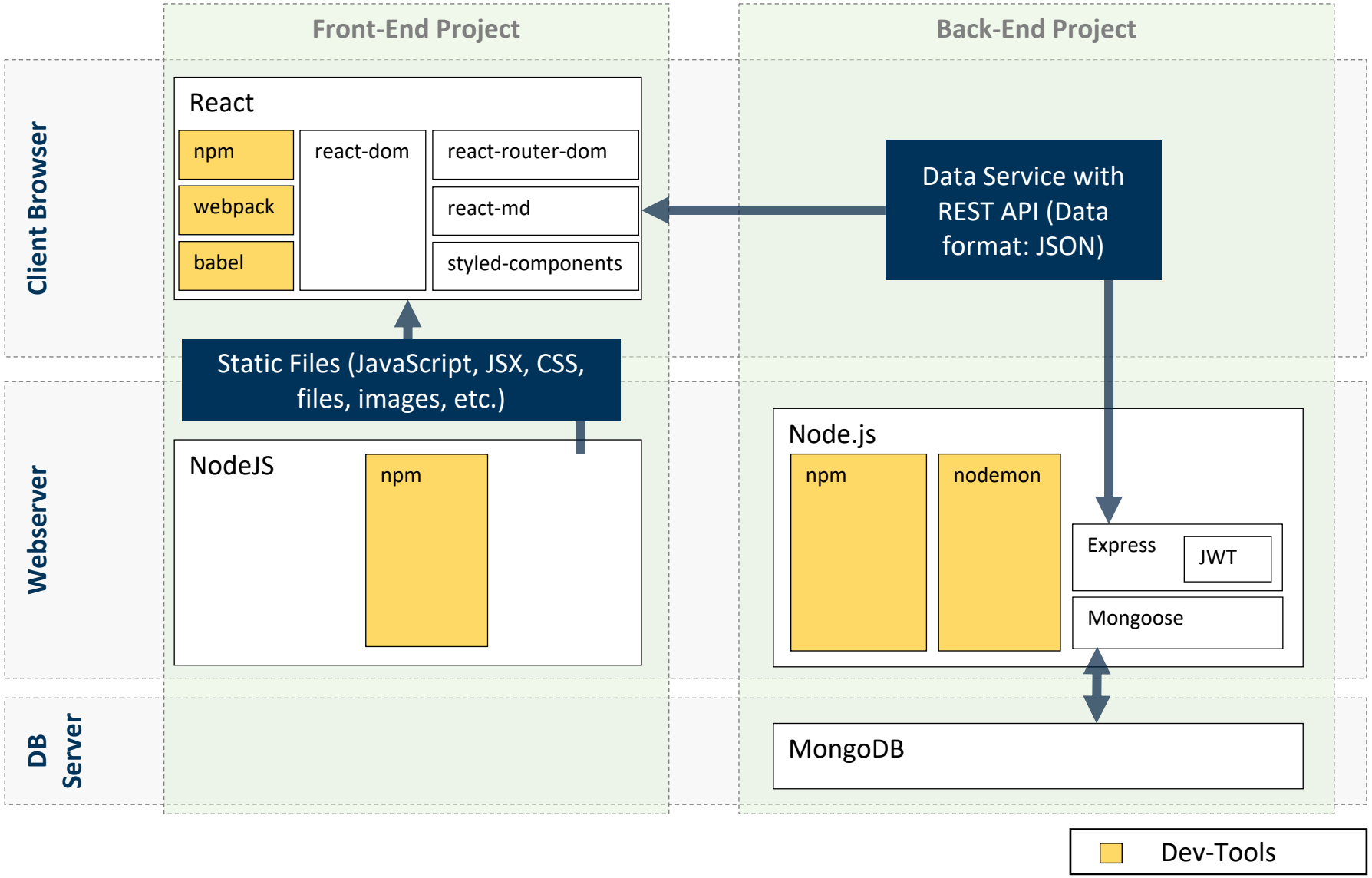
Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

Outline

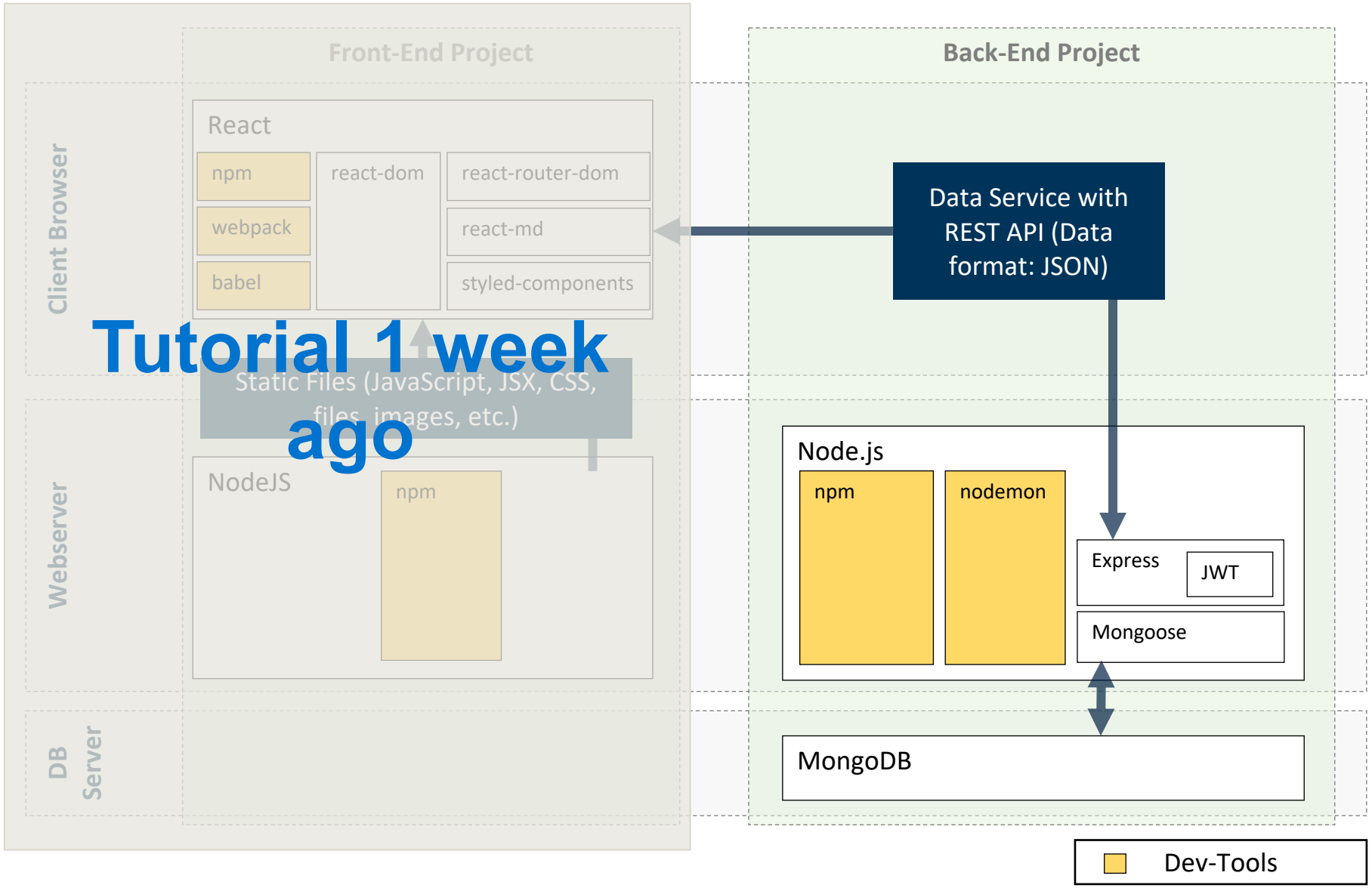
T3. Building Rest-Enabled Back-End Services

- Where are we now?
- Introduction to Node.js
- Node Package Manager (NPM)
- REST APIs with Express
- MongoDB
- Authentication & Authorization
- Structure of the Movie Backend Application

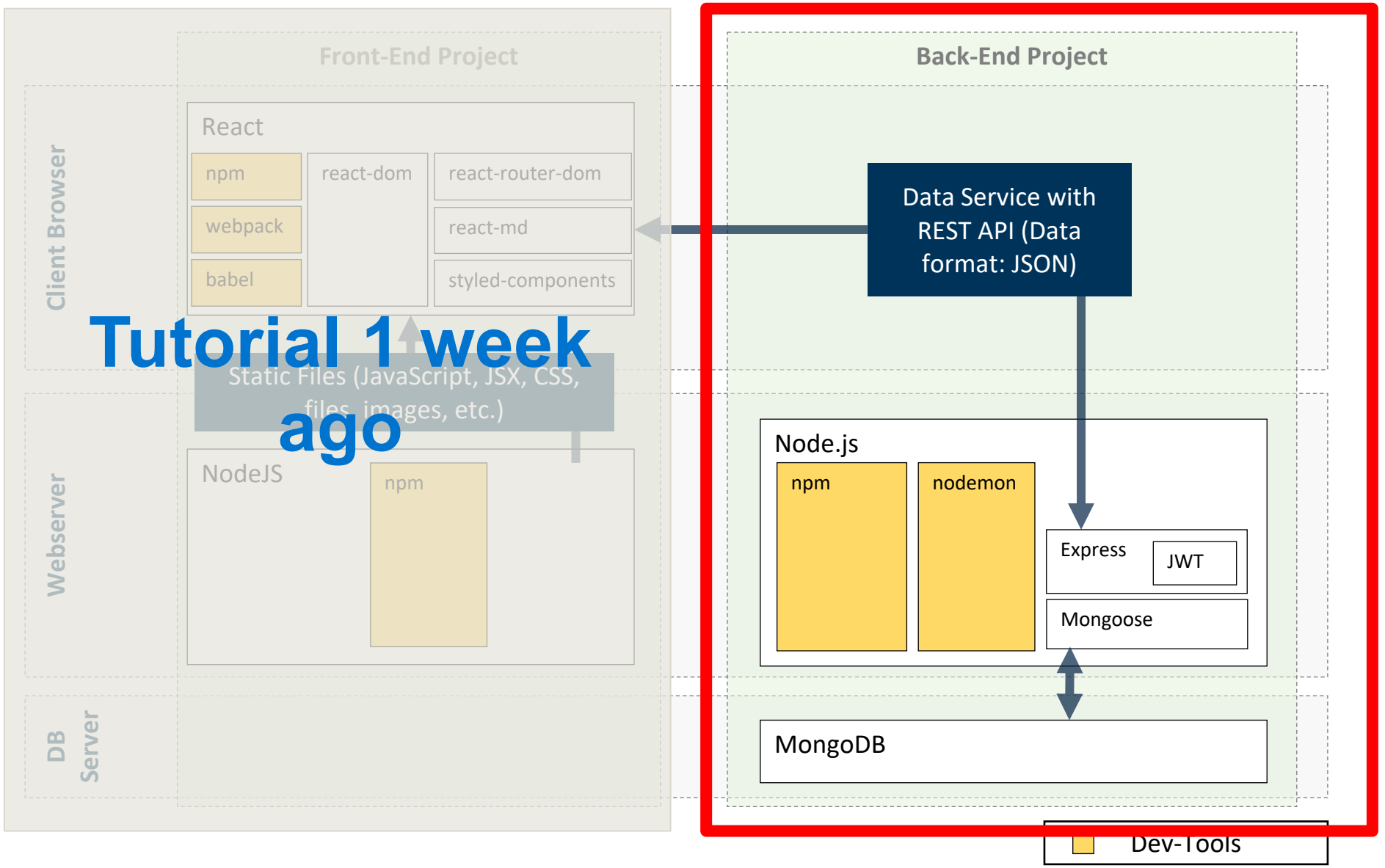
Movie Application Overview



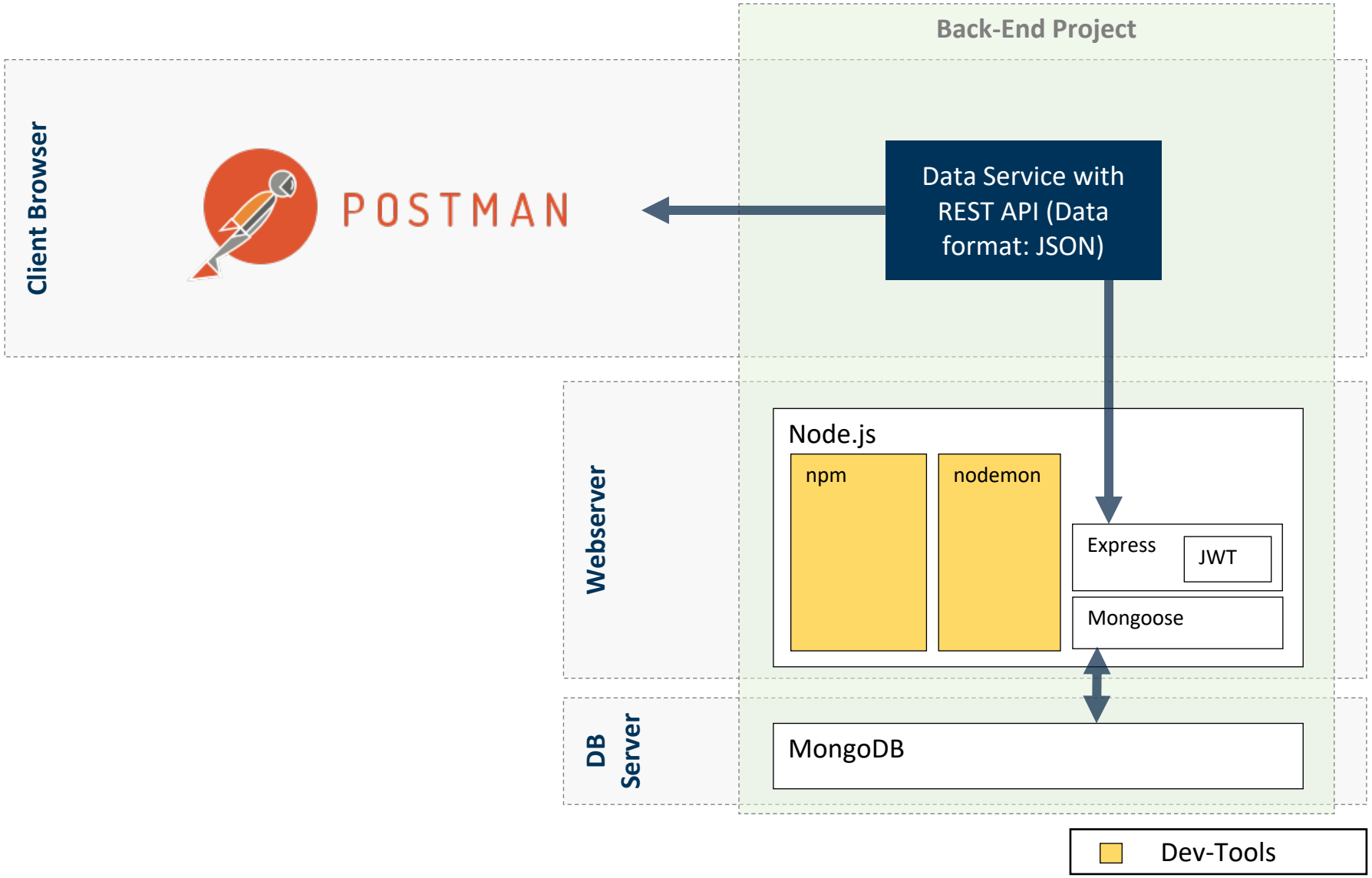
Movie Application Overview



Movie Application Overview



Movie Application Overview

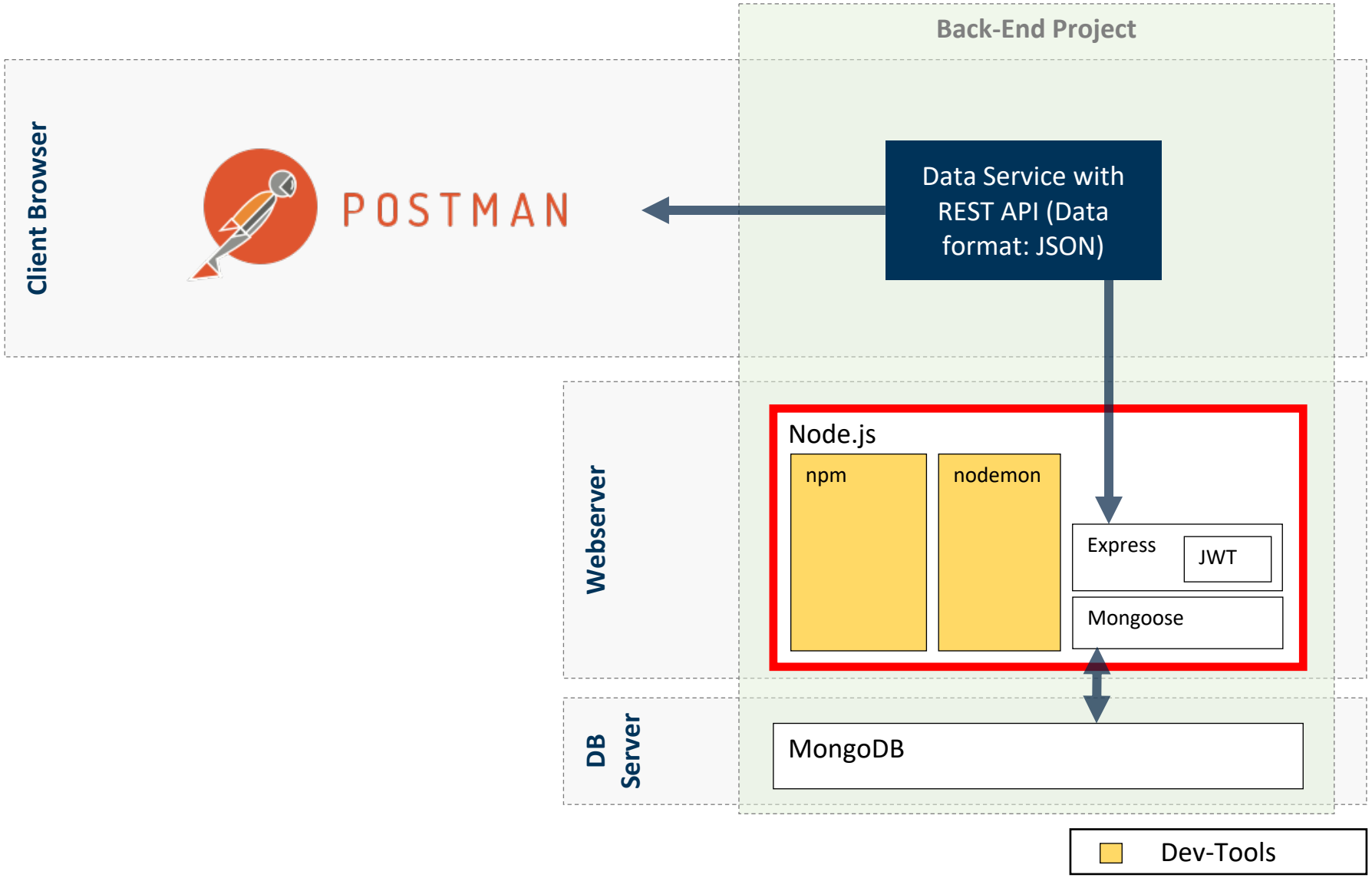


Outline

T3. Building Rest-Enabled Back-End Services

- Where are we yet?
- Introduction to Node.js
- Node Package Manager (NPM)
- REST APIs with Express
- MongoDB
- Authentication & Authorization
- Structure of the Movie Backend Application

Movie Application Overview



Introduction to Node.js

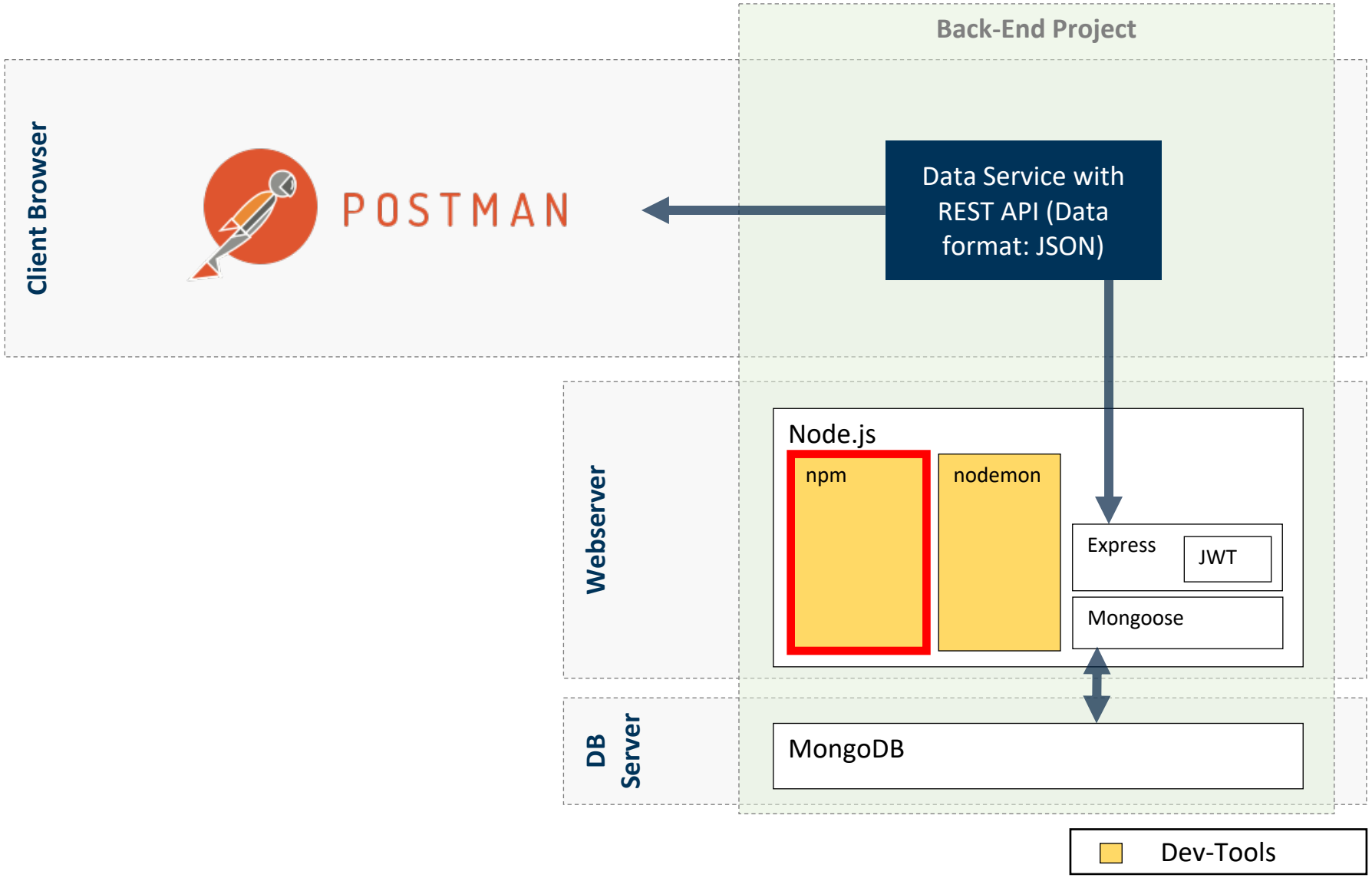
- JavaScript runtime running on top of Google's open-source JavaScript engine called V8
- Pairing JavaScript's event-driven, asynchronous coding style with non-blocking I/O libraries
- Therefore no multithreading, every request does **not** create a new thread
- The I/O loop is single threaded
- Driving forces:
  - JavaScript for frontend and backend → a single language for the whole web development process
  - Performance
  - Concurrency control

Outline

T3. Building Rest-Enabled Back-End Services

- Where are we yet?
- Introduction to Node.js
- Node Package Manager (NPM)
- REST APIs with Express
- MongoDB
- Authentication & Authorization
- Structure of the Movie Backend Application

Node Package Manager (NPM)



NPM

- Command line based package manager for handling dependencies, versions, and project properties of Node.js projects

- Provides three principal components
  - Dependency specification
  - Dependency tool
  - Repository

- Definition of project properties and dependencies in *package.json* file

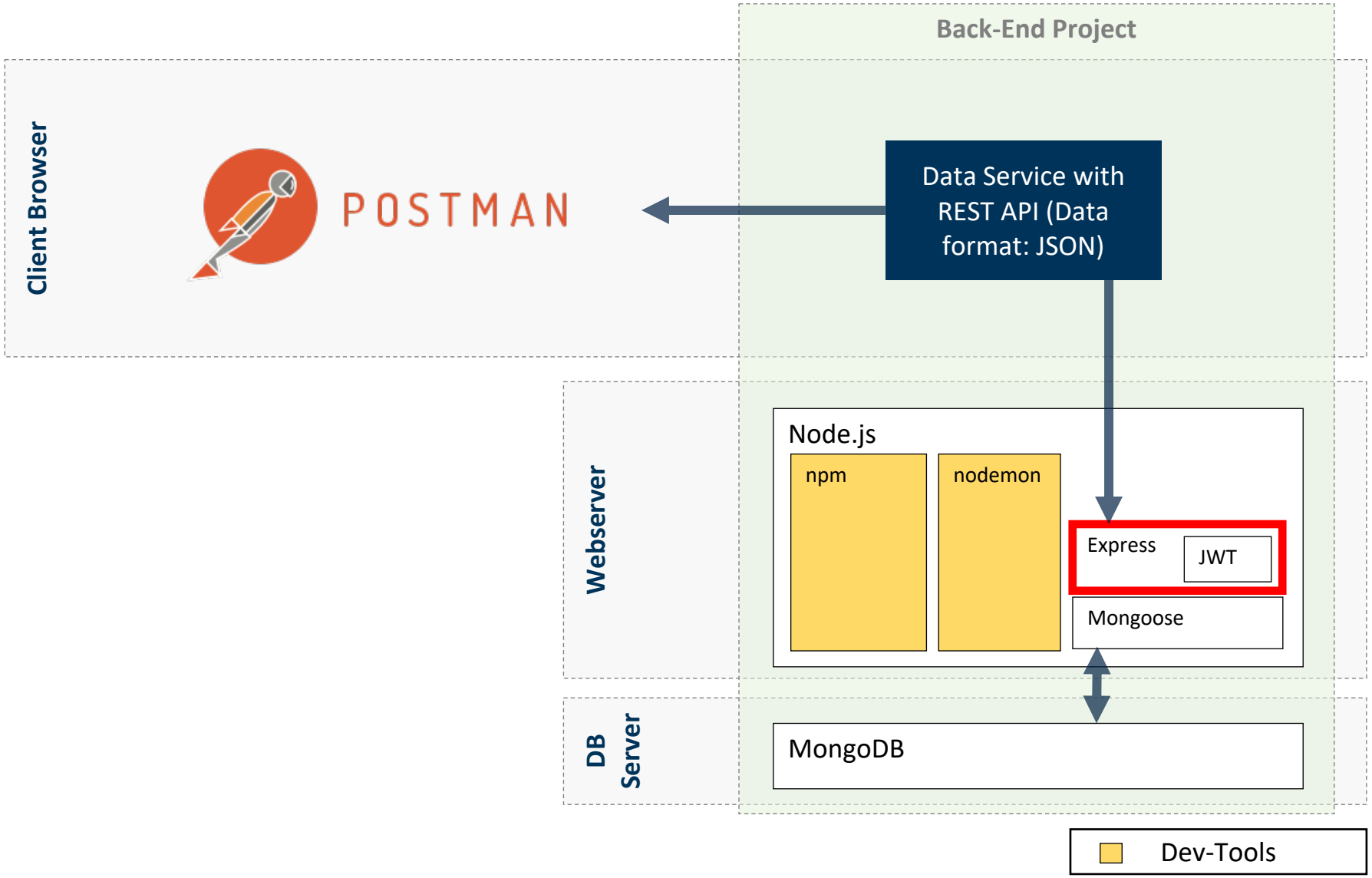
- Some commands are e.g.
  - *npm install* – install all dependencies from *package.json*
  - *npm install <package_name> --save* – adds new npm package and saves it as a dependency in *package.json*
  - See https://docs.npmjs.com/ for a comprehensive overview

Outline

T3. Building Rest-Enabled Back-End Services

- Where are we yet?
- Introduction to Node.js
- Node Package Manager (NPM)
- REST APIs with Express
- MongoDB
- Authentication & Authorization
- Structure of the Movie Backend Application

REST APIs with Express



REST APIs with Express

Express Framework

- Minimalist web application framework that supports the creation of HTTP-based (REST) APIs
- Equivalent of Java JAX-RS
- Provides a bunch of HTTP utility methods
- Takes care of routing requests
- Introduces the concept of middleware and provides existing middleware functionality
- Mimics Remote Procedure Call style communication
- Procedure:
  - Accepting the request and storing it in memory
  - Performing routing
  - Incorporating middleware functionalities
  - Handling the request
  - Producing the json response

REST APIs with Express

REST API Overview

Http Verb (CRUD operation)	URL	Authentication	expected request body	expected response
GET (READ)	/movies	No	[empty]	serialized movie object
GET (READ)	/movies/{movieId}	No	[empty]	array of serialized movie objects
POST (CREATE)	/movies	Yes	serialized movie object	serialized movie object
PUT (UPDATE)	/movies/{movieId}	Yes	serialized movie object	serialized movie object
DELETE (DELETE)	/movies/{movieId}	Yes	[empty]	[empty]

REST APIs with Express

REST API Overview

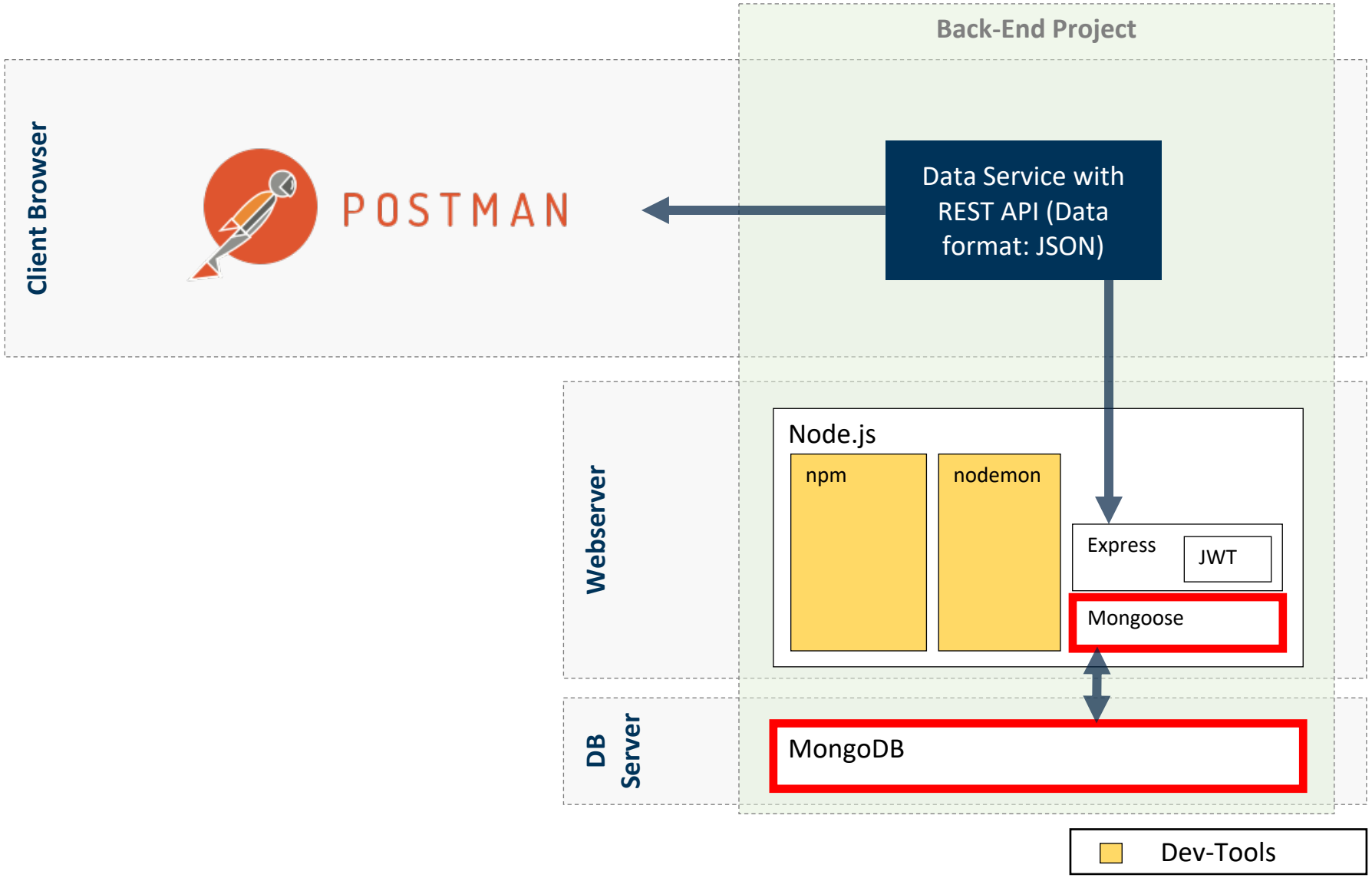
Http Verb (CRUD operation)	URL	Authentication	expected request body	expected response
GET (READ)	/auth/me	YES	[empty]	serialized user object
GET (READ)	/auth/logout	YES	[empty]	JSON object with token as null
POST	/auth/login	NO	serialized user credentials	JSON object with JSON Web Token
POST (CREATE)	/auth/register	NO	serialized user object	JSON object with JSON Web Token

Outline

T3. Building Rest-Enabled Back-End Services

- Where are we yet?
- Introduction to Node.js
- Node Package Manager (NPM)
- REST APIs with Express
- Database
- Authentication & Authorization
- Structure of the Movie Backend Application

Database



Database

MongoDB

- **Document oriented** NoSQL-Database
- **Collections** of documents
- **Collections** do not have a fixed scheme
- Scheme definitions only in the code and independent from database
- **Documents** have some built-in properties as _*id*_ (unique identifier) and _*v*_ (version number)

Database

Model Definitions with Mongoose

Movie
_id: ObjectId title: String synopsis: String runtime: Number mpaa_rating: String year: Number posters: { thumbnail: String, … }
find (object conditions, function cb) : Query findById (number id, function cb) : Query update (object conditions, object updates) : Query ... [many more] ...

User
_id: ObjectId username: String password: String
find (object conditions, function cb) : Query ... [many more] ...

Database

Mongoose - CRUD operations

- **CREATE**
  - const newMovie = new Movie({title: “Movie”, year: 2010});
  - newMovie.save(err => {})

- **READ**
  - find((err, movies) => {})
  - find({title: ”Movie”, year: 2010}, (err, movies) => {})
  - findOne({title: ”Movie”, year: 2010}, (err, movie) => {})
  - findById(movieId, (err, movie) => {})

- **UPDATE**
  - findByIdAndUpdate(movieId, {synopsis: “New synopsis”}, (err, movie) => {})
  - findOneAndUpdate({title: ”Movie”, year: 2010}, {synopsis: “New synopsis”}, (err, movie) => {})

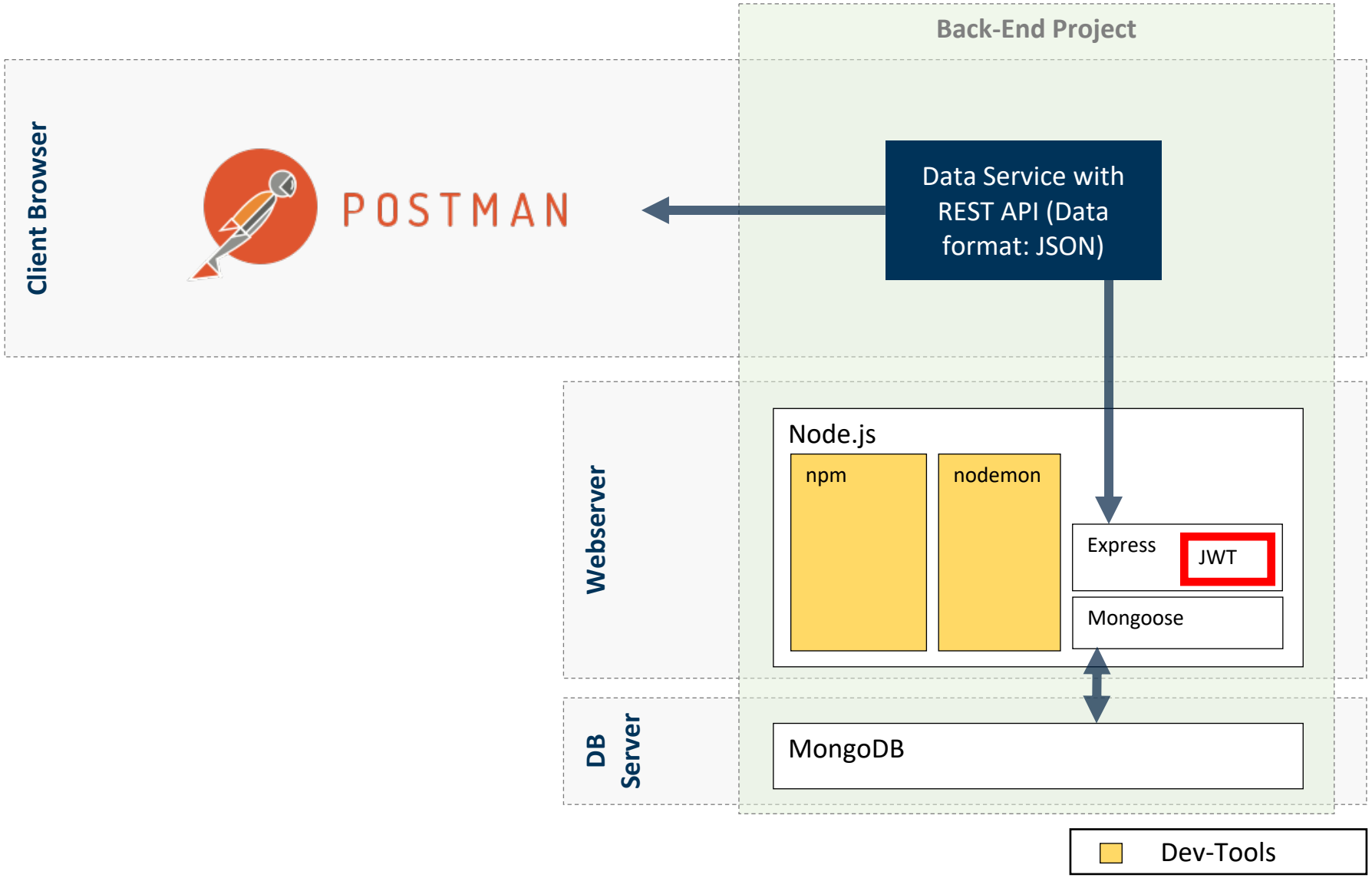
- **DELETE**
  - findByIdAndRemove(movieId, (err, movie) => {})
  - findOneAndRemove({title: ”Movie”, year: 2010}, (err, movie) => {})

Outline

T3. Building Rest-Enabled Back-End Services

- Where are we yet?
- Introduction to Node.js
- Node Package Manager (NPM)
- REST APIs with Express
- MongoDB
- Authentication & Authorization
- Structure of the Movie Backend Application

Authentication & Authorization

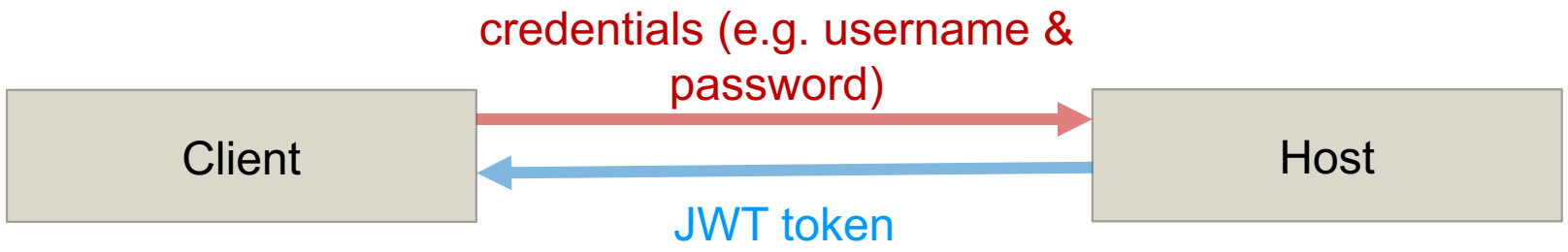


Authentication & Authorization

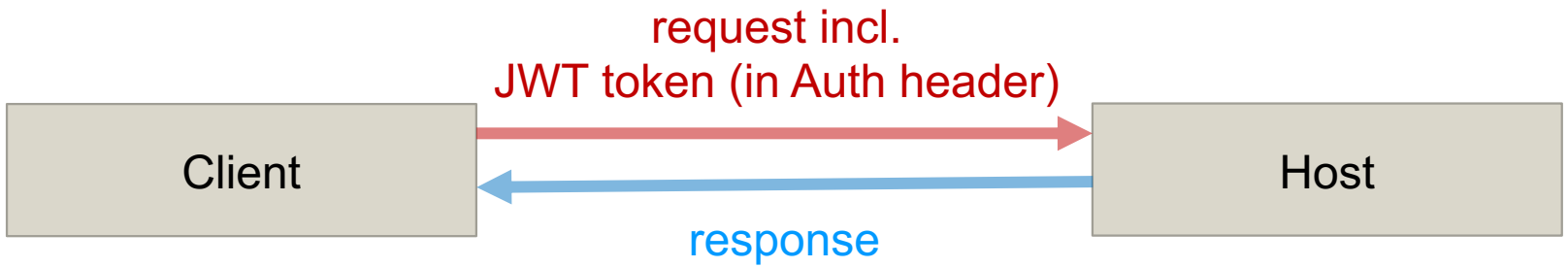
JWT

- JWT – JavaScript web tokens

- Login – client sends credentials and receives JWT



- Subsequent requests – client sends JWT in authorization header



Authentication & Authorization

JWT

- Sample token:
  - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
  - header.payload.signature

- The payload can only be **encrypted** by the host, but decrypted by everyone

- It's save to rely on payload information e.g. the user's id

- Header contains meta information such as the algorithm used for encryption

- JWT allows to use sessions in stateless paradigms as REST

Authentication & Authorization

Utilizing JWT

- Implementation as **middleware**

- Middleware can be added to certain routes and executes independently before calling actual route processing function.

- Examples of other middleware:
  - json **body-parser**: parses JSON-formated request
  - **cors**: Modifies headers to accept cross-server requests

- The middleware function parses the JWT token from the auth header.

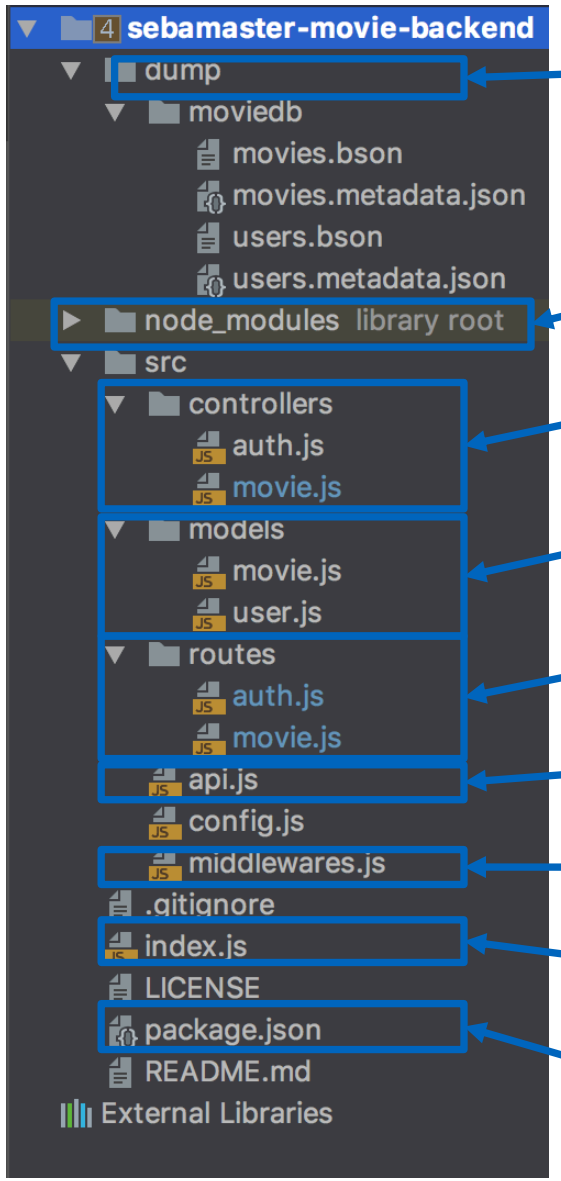
- If JWT is valid → Injects user to the request payload → Ready for e.g. further authorization

Outline

T3. Building Rest-Enabled Back-End Services

- Where are we yet?
- Introduction to Node.js
- Node Package Manager (NPM)
- REST APIs with Express
- MongoDB
- Authentication & Authorization
- Structure of the Movie Backend Application

Movie Backend File & Folder Structure



Dump of the moviedb database. There are two files for each collection, a schema definition and the actual data.

NodeJS / NPM depencies are stored in the node_modules folder.

In the controllers sub-folder all controllers are implemented for the actual request processing.

The MongoDB schema definitions take place in the models sub-folder.

The routing is defined in the routes sub-folder. Each collection defines it own routes.

The api.js is used to setup the express.js framework and mainly configures the routing.

Middleware.js defines the middleware such as authentication.

The index.js is the root source file for a node.js application. Here it contains mostly the database connection and starts the server.

The application and dependencies are configured in the package.json file.

M.Sc.

Ingo Glaser

Technische Universität München
Faculty of Informatics
Chair of Software Engineering for Business Information Systems

Boltzmannstraße 3
85748 Garching bei München

Tel +49.89.289.17138
Fax +49.89.289.17136

ingo.glaser@tum.de
wwwmatthes.in.tum.de