**Node-Express-Mongo Theory**

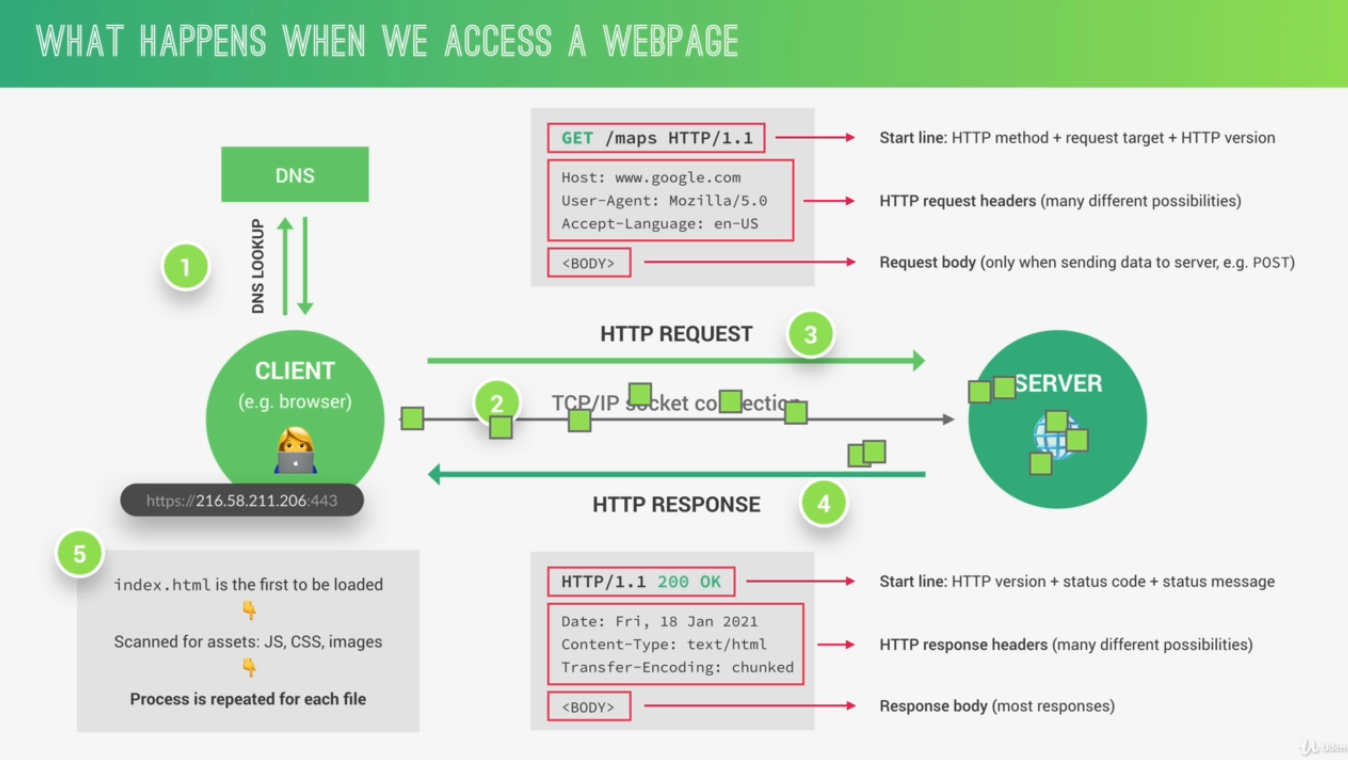
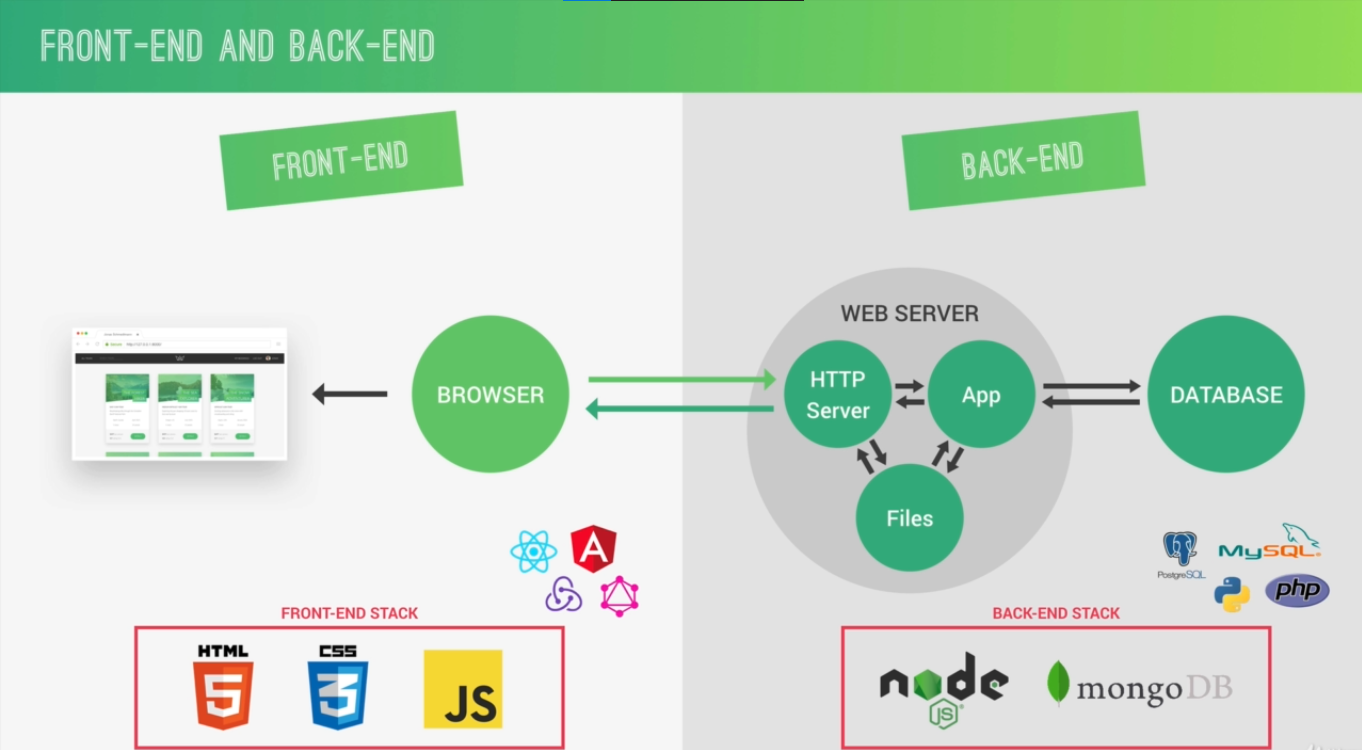
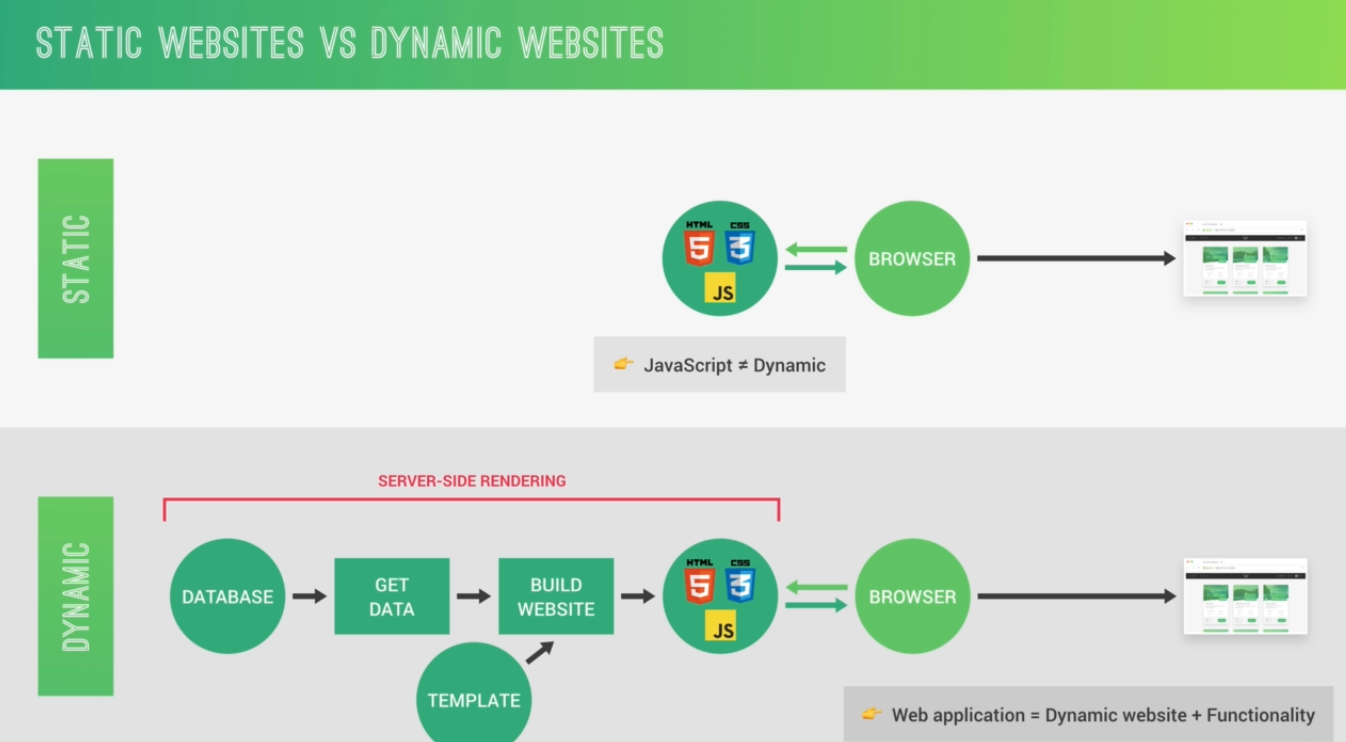
1. **Node.JS** is a JavaScript runtime built on Google’s open source V8 JavaScript engine.
2. Node.JS pros:
   1. Single-threaded, based on event driven, non-blocking I/O model
   2. Perfect for building fast and scalable data-intensive apps;
   3. Some big companies uses node in production(Netflix, uber, PayPall, ebay…)
   4. JavaScript across the entire stack: faster and more efficient development;
   5. NPM: huge library of open-source packages available for everyone for free
   6. Very active developer community
3. Use Node.JS for:

- API with database behind it (preferably NoSQL);

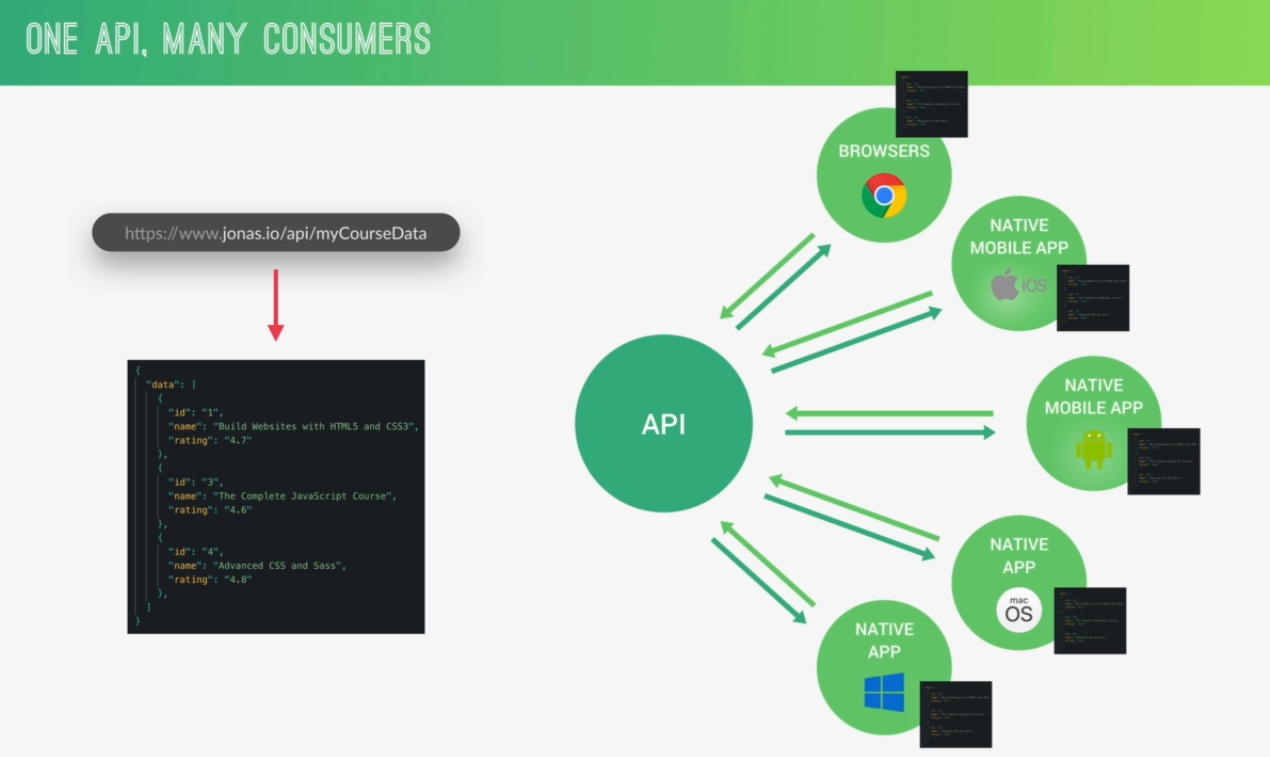
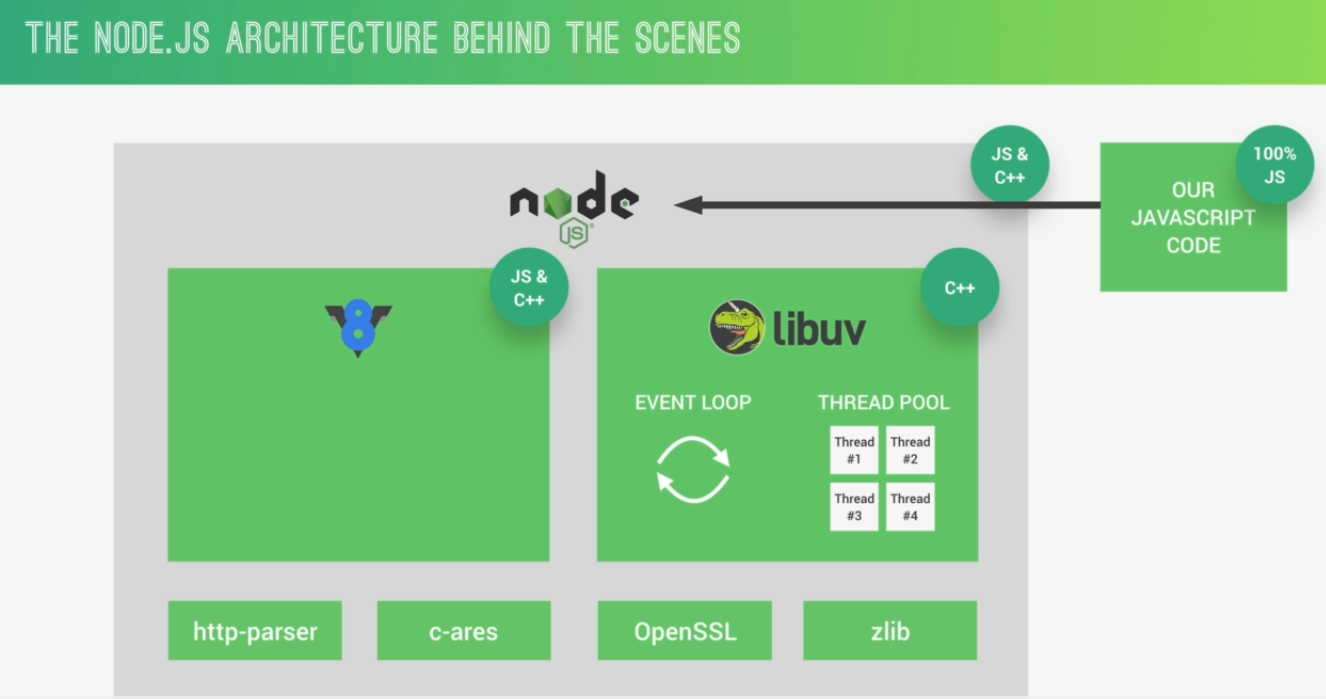
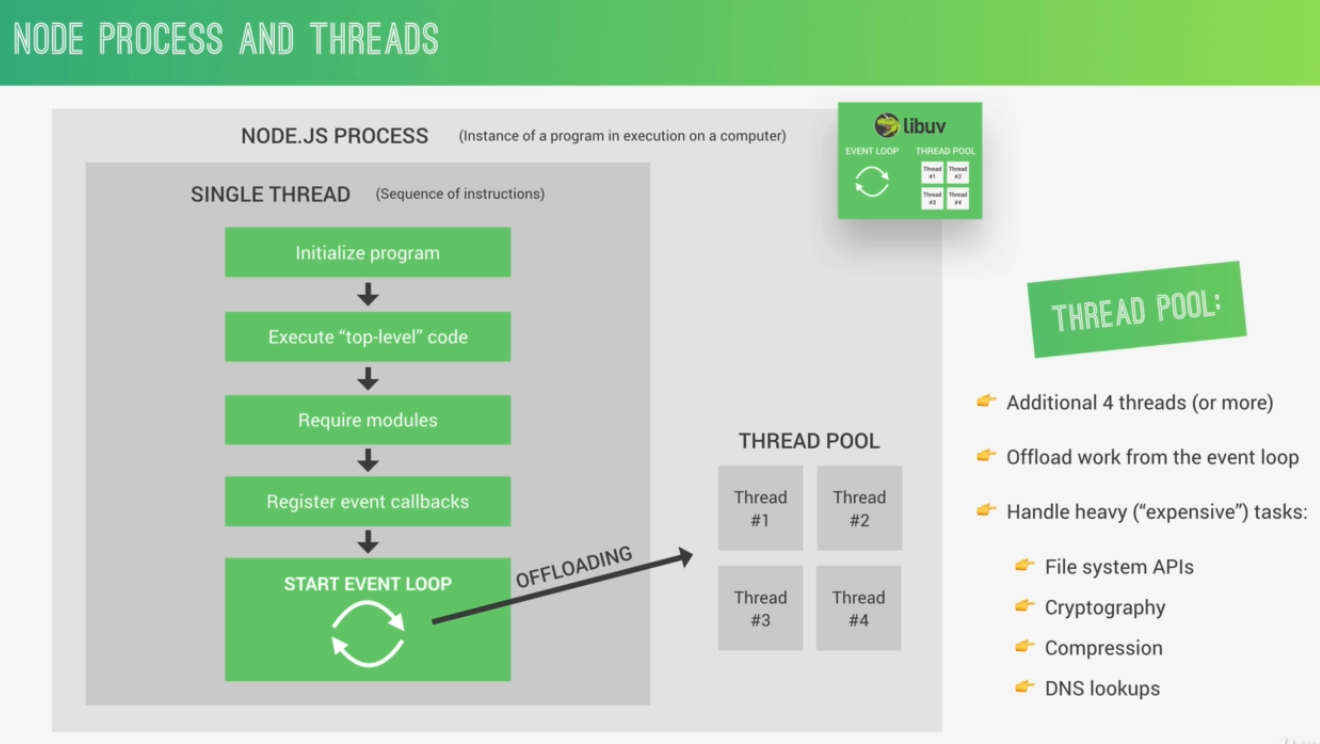
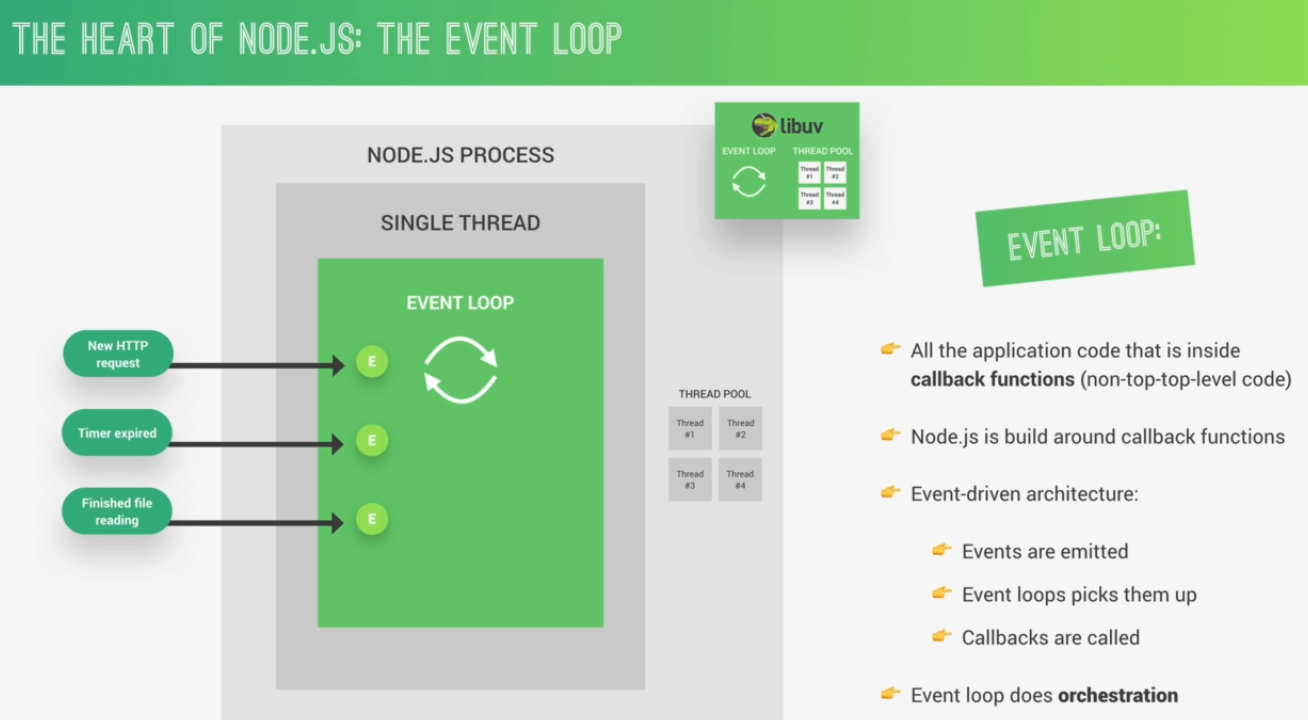
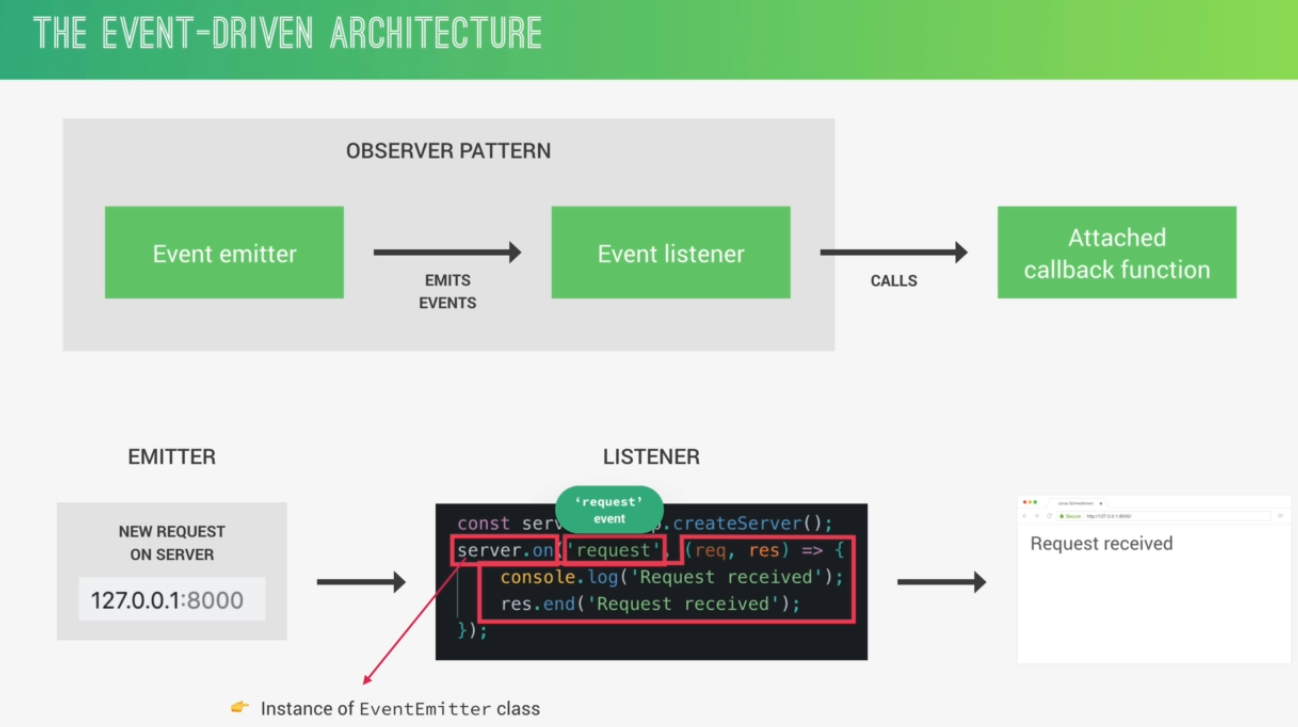
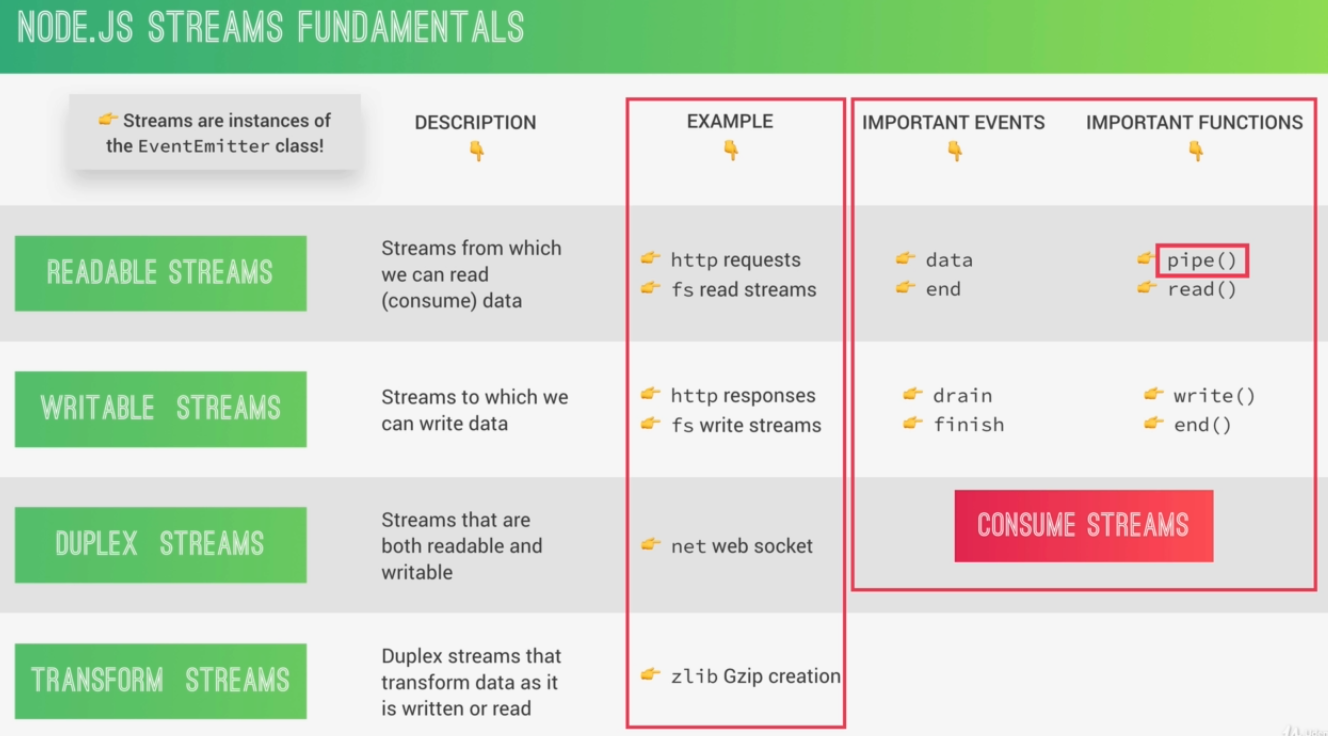
- Data streaming (think Youtube);

- Real-time chat application

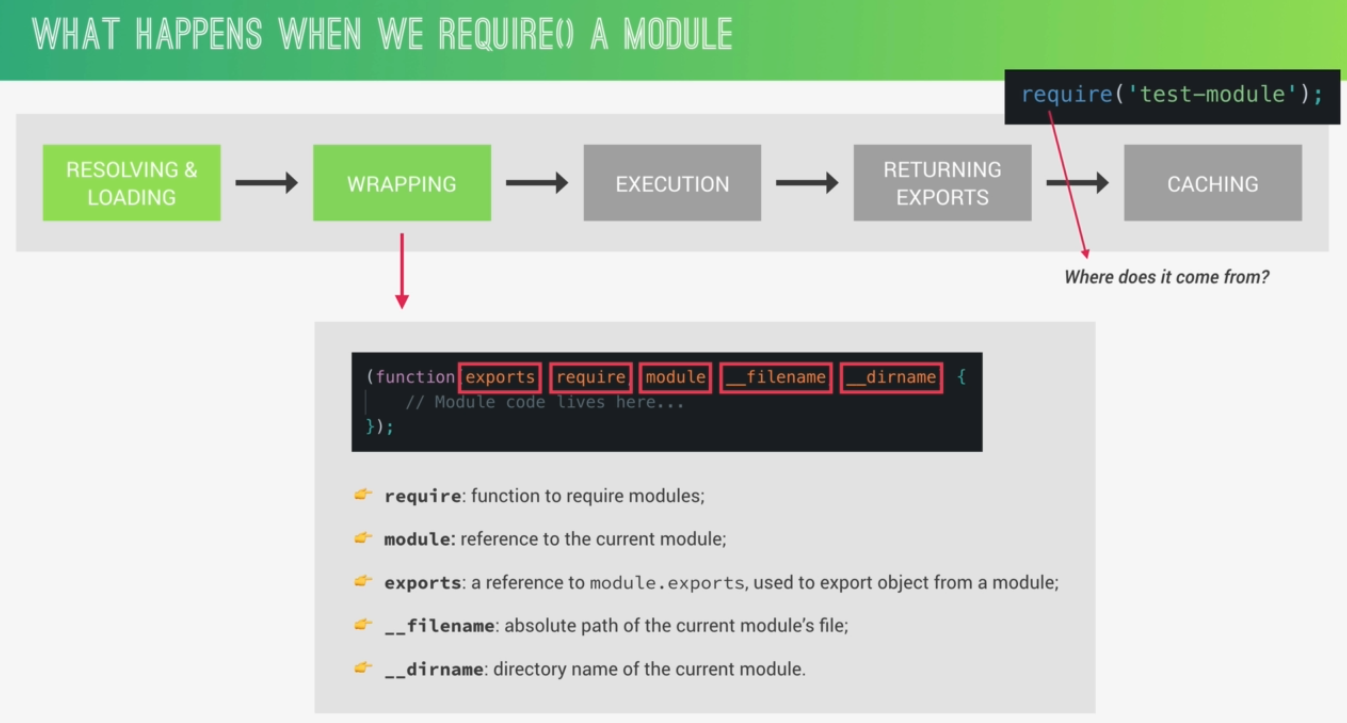
-Server-side web application

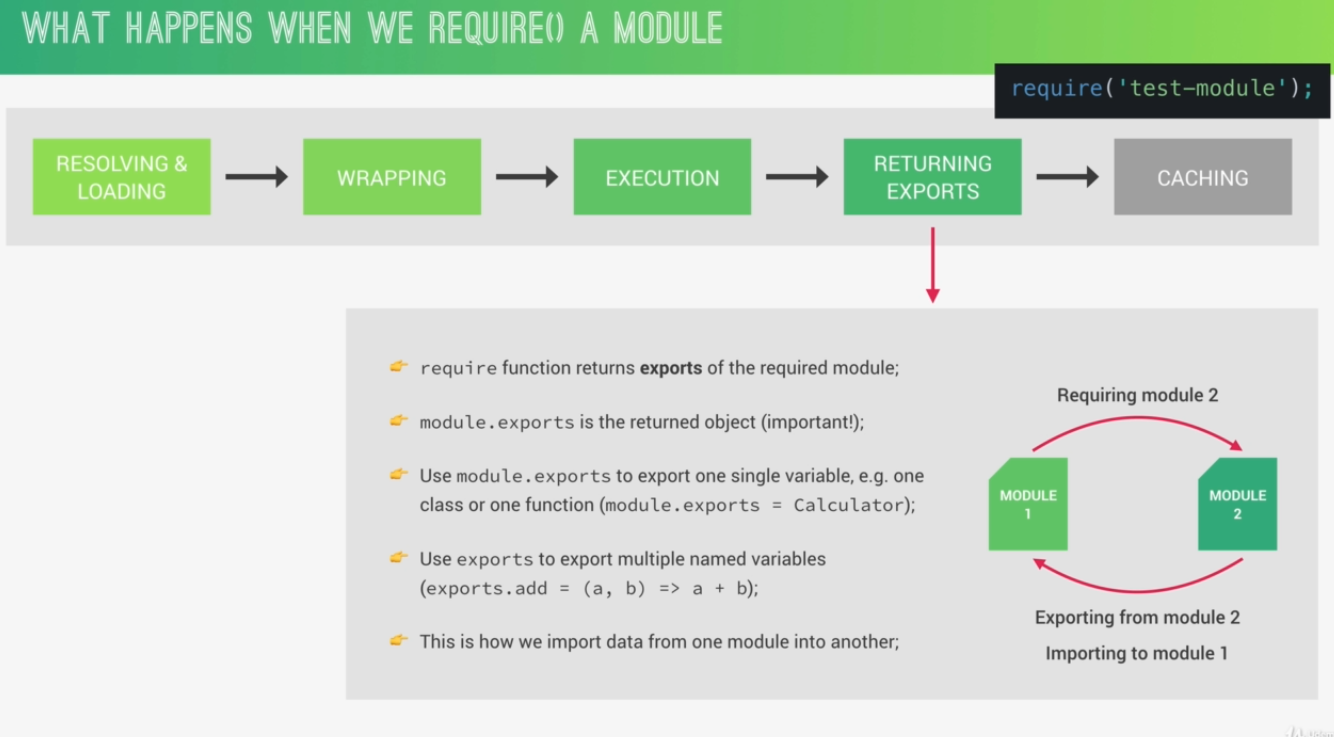
1. Don’t use Node.JS for the heavy server-side processing (CPU intensive) like image processing, video manipulation, file compressing etc.
2. Node is Single Threaded so if you use synchronous code it means that any other operation needs to wait for the first one to be finished
3. Use Asynchronous functions for executing operations. Using callback function in Node.JS
4. To avoid callback hell we can use Promises or Async/Await
5. Dependencies are of two types: regular dependencies where the code depends on the dependency and the dev-dependencies that help the developers with tools and helpers for the developers(like webpack, nodemon etc.). Nodemon is good for listening for the changes and restarting the server for us instead of using “node index.js”.
6. If we want to use a local script in the terminal we need to declare them into the scripts. We can only use global dependencies directly in the terminal
7. What is a “slug”? It is a part of a URL that identifies a particular page on a website in an easy-to-read form. A slug is the part of the URL that explains the page’s content. (slugify used)
8. Code for dependencies versions. Ex: \*^~1.18.10 (1 is the major version, 18 is the minor version, and the 10 is the patch version). The “~” stands for only patch releases(which is safer), the “^” is for all the patch and minor releases. There is also an option to update to all version using “\*”
9. Util commands : npm outdated, npm update(that doesn’t work updating the package.json, but package-lock is working in npm 6+), npm I package@version, npm update <packageName>, npm -rm -r node\_modules, npm uninstall <packageName>, npm install
10. Request-response architecture
11. Frontend-backend
12. Static vs dynamic websites
13. Dynamic vs Api-powered websites



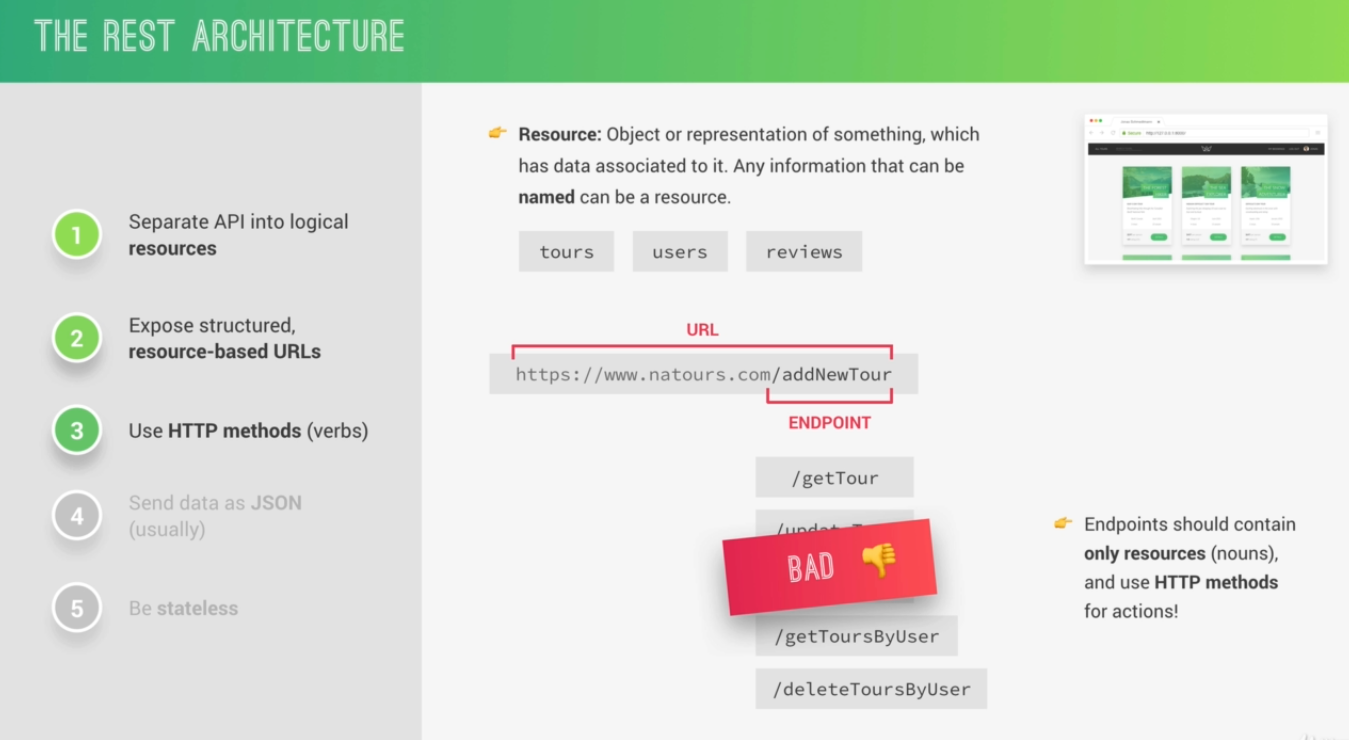
1. Dynamic websites are computing the data on the serverside and the website is build there based on the actions the users makes. The server makes the pages and send them back to the browser to be displayed. The Api powered website is getting the data from a database, and expose it to the frontend through JSON and the browser gets the data and you can build the website on the frontend alone. You can make an API server and it can exist alone to be consumed by different consumers.
2. Api severs 
3. Node is a JS runtime which uses “V8” dependency powered by google to translate the JavaScript into machine code. But is not done alone, it needs also “libuv” open source library with a strong focus on asynchronous I/O. This layer give V8 the capacity to access the underling operation system with the files system, networking etc. The libuv implements two important features: The even loop(executing tasks and network I/O) and the Thread pool(for heavy work: file compression , file access). Libuv is written in C++ and the V8 is written in JS with C++.
4. The node js is having a single process with a single thread. You need to be careful to not block the thread. When the work is to much to the even loop it handles tasks for the thread pool
5. The thread pool takes care of the heavy work:
   1. has 4 aditional threads(or can be setted more)
   2. Offload work from the even loop
   3. Handle heavy(“expensive”) tasks like: file system APIs, Cryptography, Compression, DNS lookups(matching the server domain with an address)
6. Even loop is the heart of the node architecture:
   1. All the application code that is inside callback functions(non-top-top-level code)
   2. Node.js is build around callback functions
   3. Even-driven architecture: -events are emitted; -event loops picks them up; - callbacks are called
   4. Even loop does the orchestration
7. Event loop has 4 phases. It needs to clear the callbacks from each phase to advance to the next phase. The first phase is the Expired timer callbacks that is handling some expire timers like setTimeout function. After the callback queue is finished than it advances to the next phase: I/O polling and callbacks(networking and file accessing-read, write). Next phase is a setImmediate callbacks to be called after the I/O polling phase is finished. And the last phase is the Close callbacks that is called if the web server is down. Other phases : - Process.nexttick()queue and Other microtasks queue( resolved promises). (this phases are executed when any main phases is finished, high priority)
8. The even loop is running if any I/O timers is working like when the file is read or written, or the the networking I/O like listening to a request
9. The event loop is what is powering the asynchronous design and makes node.js possible. Ways of NOT BLOCKING it:
   1. Don’t use sync versions of functions in fs, crypto and zlib modules in your callback functions
   2. Don’t perform complex calculation (loops inside loops)
   3. Be careful with JSON in large objects
   4. Don’t use too comple regular expressions(nested quantifiers)
   5. Are some ways to manually unblock the even loop, but it is more complicated
10. The Even-driven architecture is based on the observer pattern where we set the server with listeners and waits for requests to be made from a client. When the request is made the callback is called from the even listener and the response is computed.
11. Streams – Used to process (read and write) data piece by piece (chunks), without completing the whole read or write operation, and therefore without keeping all the data in memory.
    1. It is perfect for handling large volumes of data, for example videos (youtube, Netflix)
    2. More efficient data processing in terms of memory (no need to keep all data in memory) and time (we don’t have to wait until all the data is available)
12. There are 4 types of node.js streams: Readable streams, Writable streams, Duplex Streams, Transform Streams
13. The common Module System:
    1. Each JavaScript file is treated as a separate module;
    2. Node.js uses the CommonJs module system: require(), exports or module. Exports;
    3. ES module system is used in browsers: import/export;
    4. There have been attempts to brin ES modules to node.js(.mjs)
14. When we require a module => Resolving &Loading => Wrapping => Execution => Returning Exports => Chaching

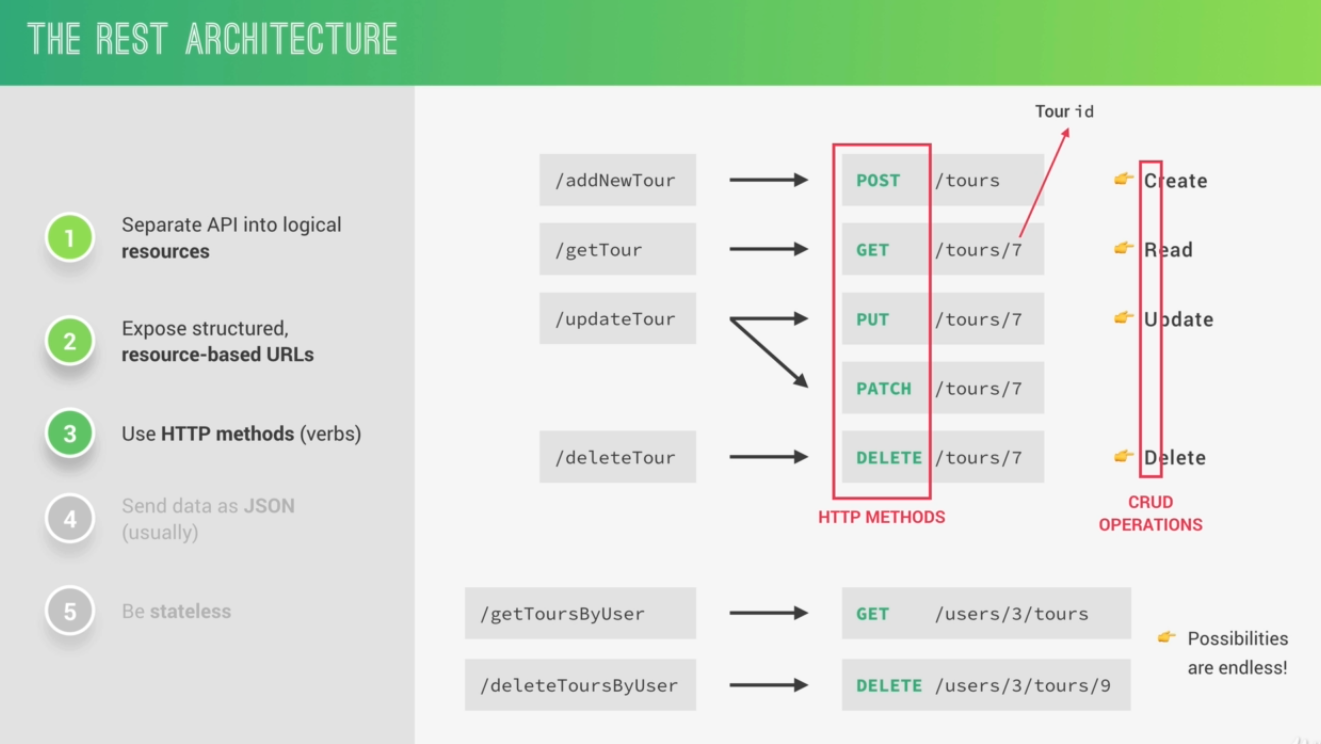


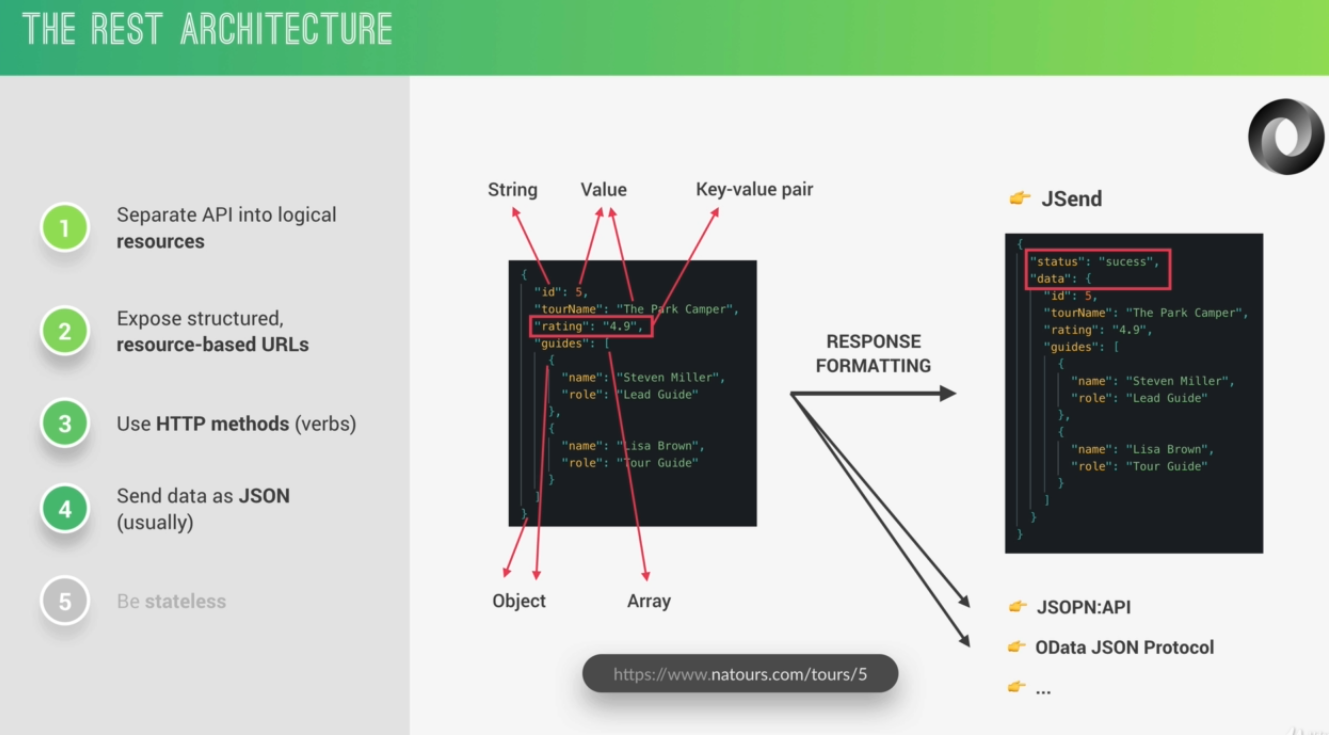


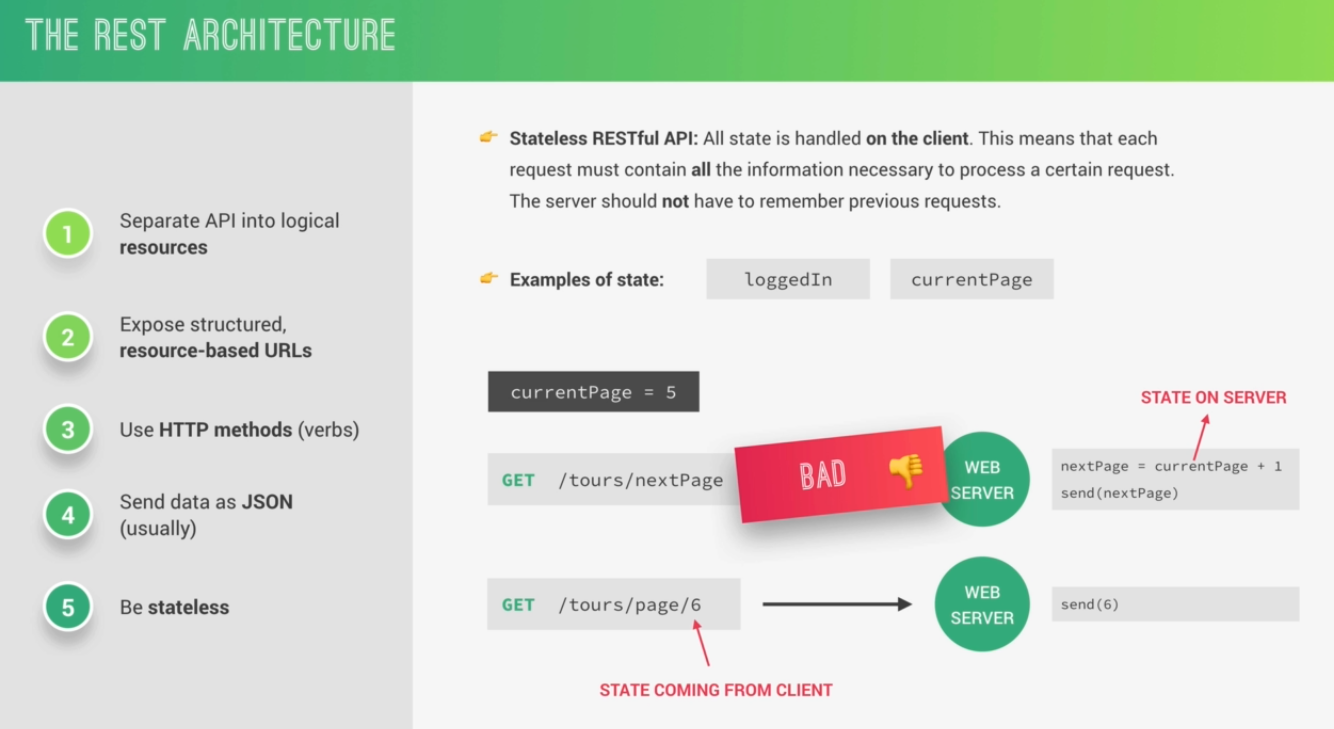


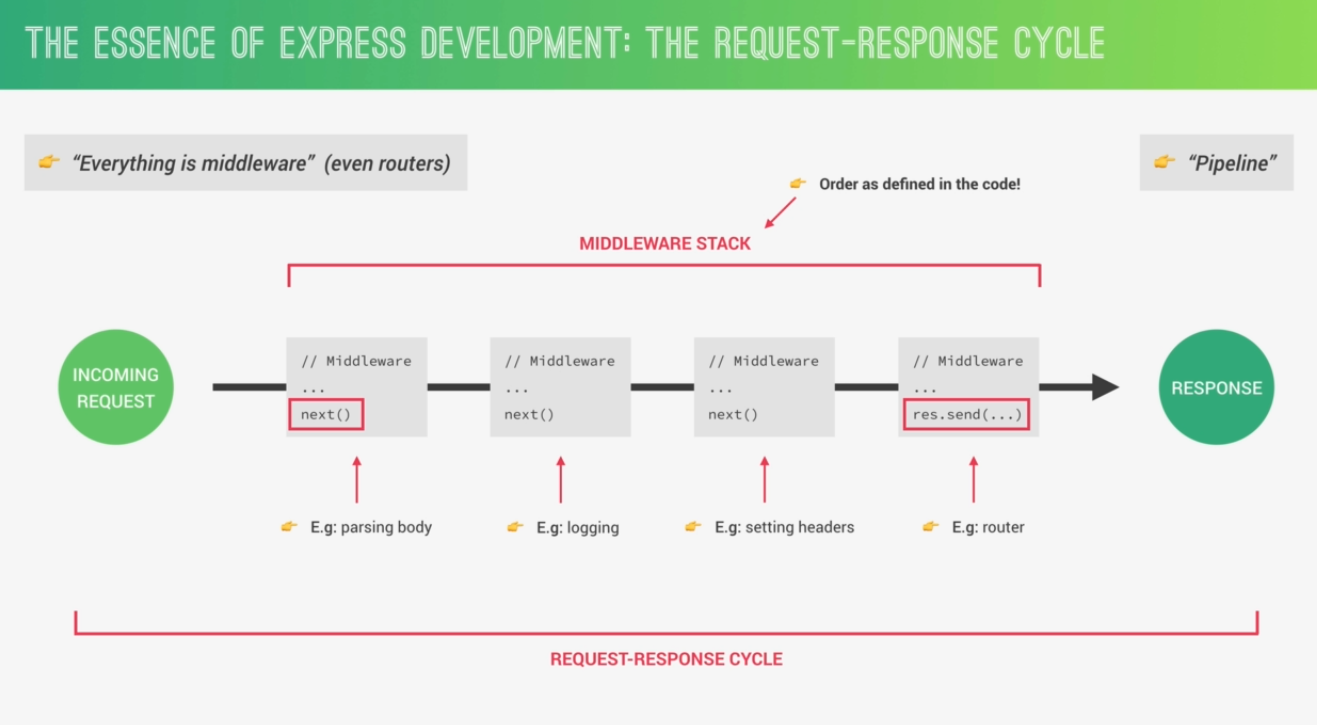
1. The await function return a promise and the return from the function will be the resolve for that promise.
2. **Express** – What it is and why use it?
   1. Express is a minimal node.js framework, a higher level of abstraction;
   2. Express contains a very robust set of features: complex routing, easier handling or requests and responses, middleware, server-side rendering, etc.
   3. Express allows for rapid development of node.js applications: we don’t have to re-invent the wheel
   4. Express makes it easier to organize our application into the MVC architecture
3. API- Application Programming Interface: a piece of software that can be used by another piece of software, in order to allow application to talk to each other.
4. Types of APIs:
   1. Web APIs
   2. Node.js “fs” or “http” APIs(“node APIs”)
   3. Browser’s DOM JavaScript API
   4. With object-oriented programming, when exposing methods to the public, we’re creating an API
5. The REST Architecture is based on some rules: 1. Separate API into logical resources; Expose structured, resource-based URLs; 3. Use HTTP methods(get, post, put, delete…); 4. Send data as JSON(usually); 5. Be stateless;
6. For communication use JSend that is enveloping the original JSON and adding to it some properties we might be interested in, like the status of the operation. Some protocols like JSOPN: API or OData JSON Protocol
7. Stateless RESTful API: All state is handled on the client. This means the each request must contain all the information necessary to process a certain request. The server should not have a to remember previous requests.



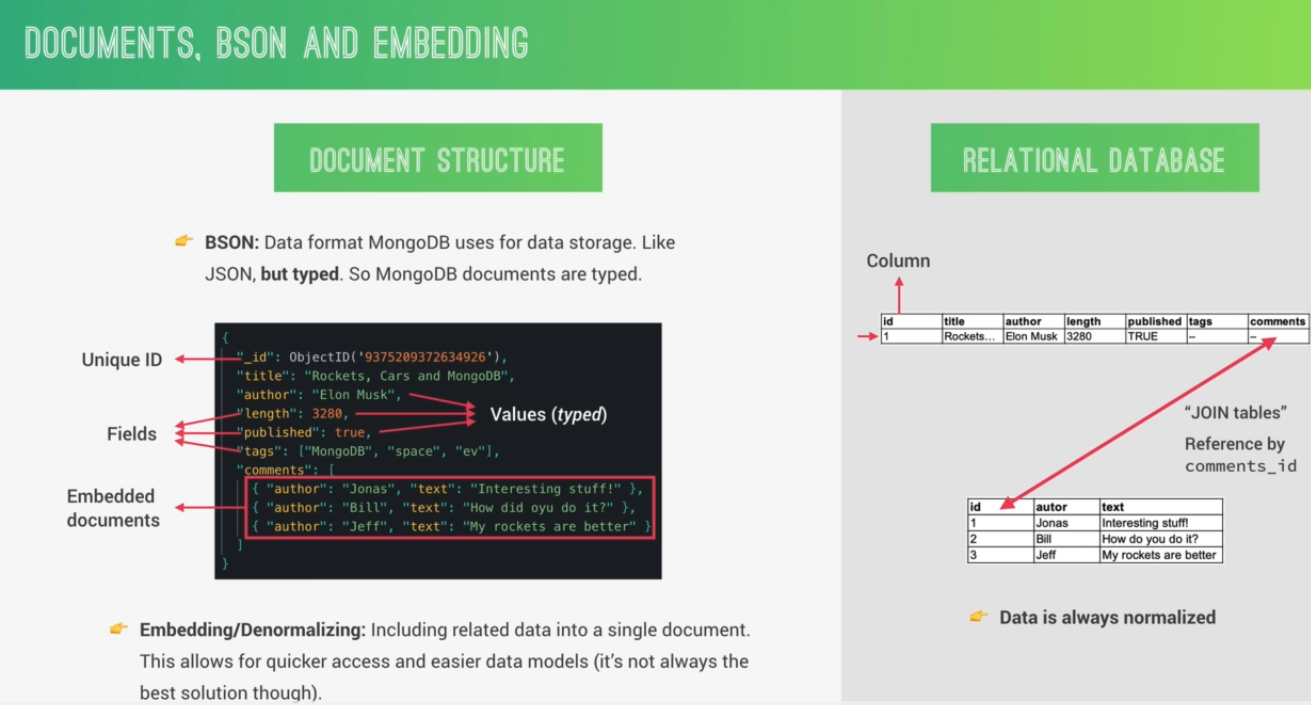






1. Codes for status => need to review (201 stands for new resource created)
2. The essence of express development: The Request-Response Cycle. The express use the middleware stack(that can be for parsing the body, logging, setting header, router…) for handling the request and send the response.
3. Middleware morgan
4. For setting the eslint with some options use “npm i eslint prettier eslint-config-prettier eslint-plugin-prettier eslint-config-airbnb eslint-plugin-node eslint-plugin import eslint-plugin-jsx-a11y eslint-plugin-react --save-dev”
5. **MongoDB** – MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need.

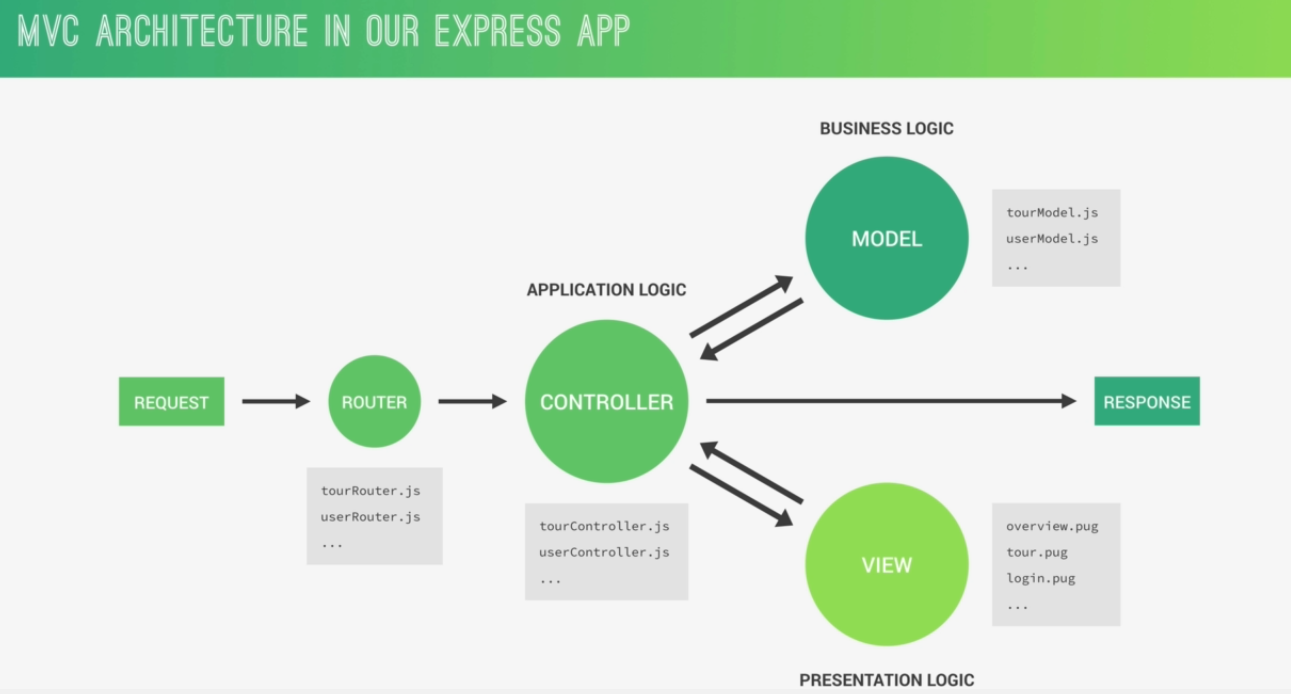


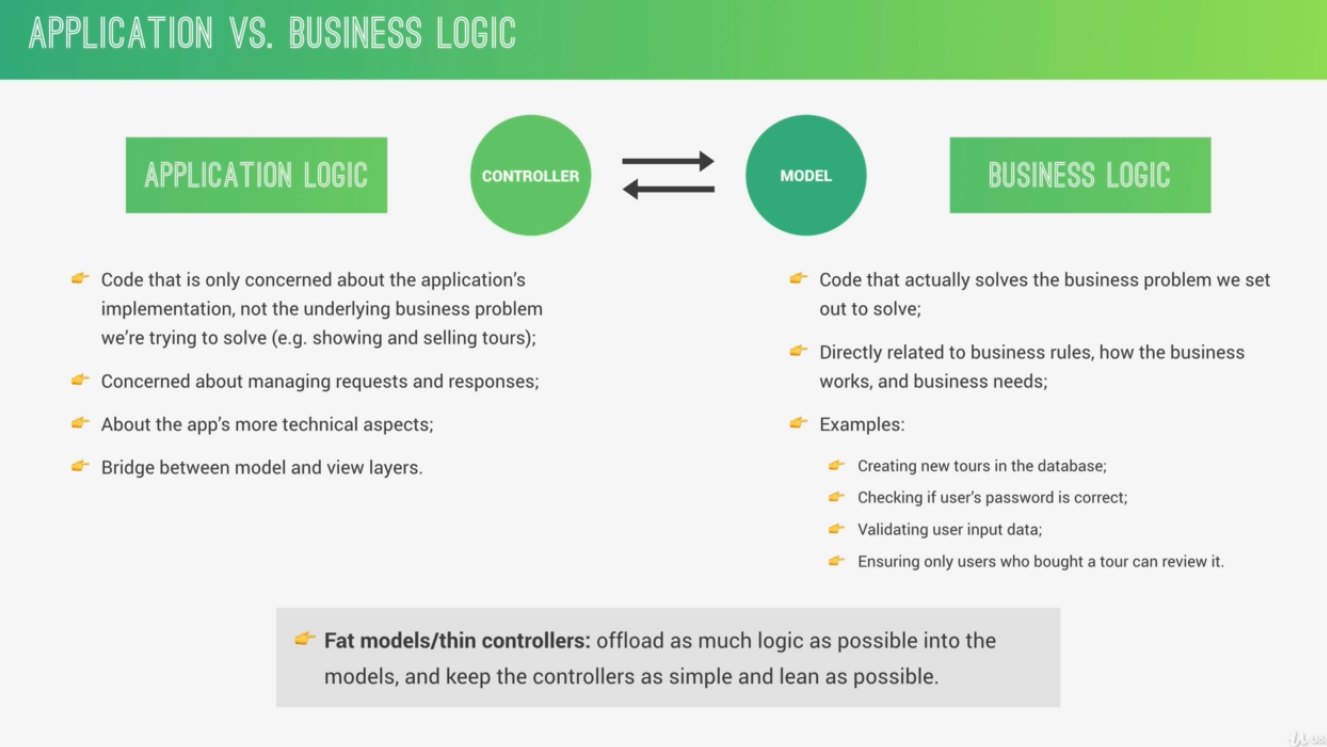


1. MongoDB key features:
   1. Document based: MongoDB stores data in documents ( field-value pair data structures, NoSQL).
   2. Scalable: Very easy to distribute data across multiple machines as your users and amount of data grows.
   3. Flexible: No document data schema required, so each document can have different number and type of fields.
   4. Performant: Embedded data models, indexing , sharding, flexible documents, native duplication, etc.
   5. Free and open-source, published under the SSPL License.
2. MongoDB document structure. BSON is a data format MongoDB uses for data storage. Like JSON, but typed. So MongoDB documents are typed.
3. Embedding/Denormalizing: Including related data into a single document. This allowas for quicker access and easeier data models (it’s not always the best solution though). This is different than the relational database where you have to define a new table if you want to have some references to other objects(tables) like comments, authors….
4. CMD commands: mongod(to start the service), mongo(to access the shell for the server), use natours-test (to create a new database or use the existing one), db.tours.insertOne({name:”The Forest Hiker”, price:297, rating:4.7}) (to insert an new bson file into the database), db.tours.find()(to find the entries from the tours bson file), show dbs(to show all the databases), use admin(to change the database), show collection(to show the created collections) ,
5. Queries: db.tours.insertMany(**CREATE**-insert multiple entires ), **READ** -> db.tours.find({name: “The Forest Hiker”}), db.tours.find({price: {$lte:500}})(mongo operator to find the objects with price lower than 500); db.tours.find({price:{$lt:500}, rating:{$lte:4.8}})(using greater or equal than, less than)-AND query; db.tours.find({$or: [ {price:{$lt:500}}, {rating:{$gte:4.8}} ]}) -> OR query; db.tours.find({$or: [ {price:{$gt:500}}, {rating:{$gte:4.8}} ]},{name:1}) (if we need to use the filter for the properties for output=> it is called project); **UPDATE** -> db.tours.updateOne({name:"The Snow Adventurer"}, {$set: {price:597}}); db.tours.updateMany({price:{$gt:500}, rating: {$gte:4.8}}, {$set: {premium:true}}) (setting new properties also); replace one by providing the find, and then the new object to replace the old one find db.tours.replaceOne(): **DELETE** -> db.tours.deleteMany({rating:{$lt:4.8}}), or deleteOne…, for deleting all use db.tours.deleteMany({});
6. We can use MongoDB Compass to manually make changed into DB
7. **MONGOOSE** :
   1. Mongoose is an Object Data Modeling(ODM) library for MongoDB and Node.js, a higher level of abstraction
   2. Mongoose allows for rapid and simple development of mongoDB database interactions
   3. Features: schemas to model data and relationships, easy data validation, simple query API, middleware
   4. Mongoose shcema: where we model our data, by describing the structure of the data, default values, and validation
   5. Mongoose model: a wrapper for the schema, providing an interface to the database for CRUD operations



1. Application vs. Business Logic





1. Application Logic:
   1. Code that is only concerned about the application’s implementation, not the underlying business problems we’re trying to solve (ex. Showing and selling tours).
   2. Concerned about managing requests and responses.
   3. About the app’s more technical aspects.
   4. Bridge between model and view layers.
2. Business Logic:
   1. Code that actually solves the business problems we set out to solve
   2. Directly related to business rules, hot the business works, and business needs
   3. Examples: Creating new tours in the database; Checking if user’s password is correct; Validating user input data; Ensuring only users who bought a tour can review it.
3. Fat models/ thin controllers: offload as much logic as possible into the models, and keep the controllers as simple and lean as possible.
4. When we use prototype then the methods will be available for all the instances of that class
5. Aggregation pipepline
6. Middleware for the model
7. For validators for mongoose we can use the “validator” library
8. Debugging with “ndb” (npm I ndb –globaly)
9. **Error Handling** –
10. Operational Errors(Exceptions)- Problems that we can predict will happen at some point, so we just need to handle them in advance.
    1. Invalid path accessed.
    2. Invalid user input (validator error from mongoose).
    3. Failed to connect to server.
    4. Failed to connect to database.
    5. Request timeout etc.
11. Programming errors- Bugs that we developers introduce into our code. Difficult to find and handle.
    1. Reading properties on undefined.
    2. Passing a number where an object is expected.
    3. Using await without async.
    4. Using req.query instead of req.body etc.
12. 