CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

DAT490 Fullstack Web Development

Dr. Philipp Leitner

philipp.leitner@chalmers.se

@xLeitix

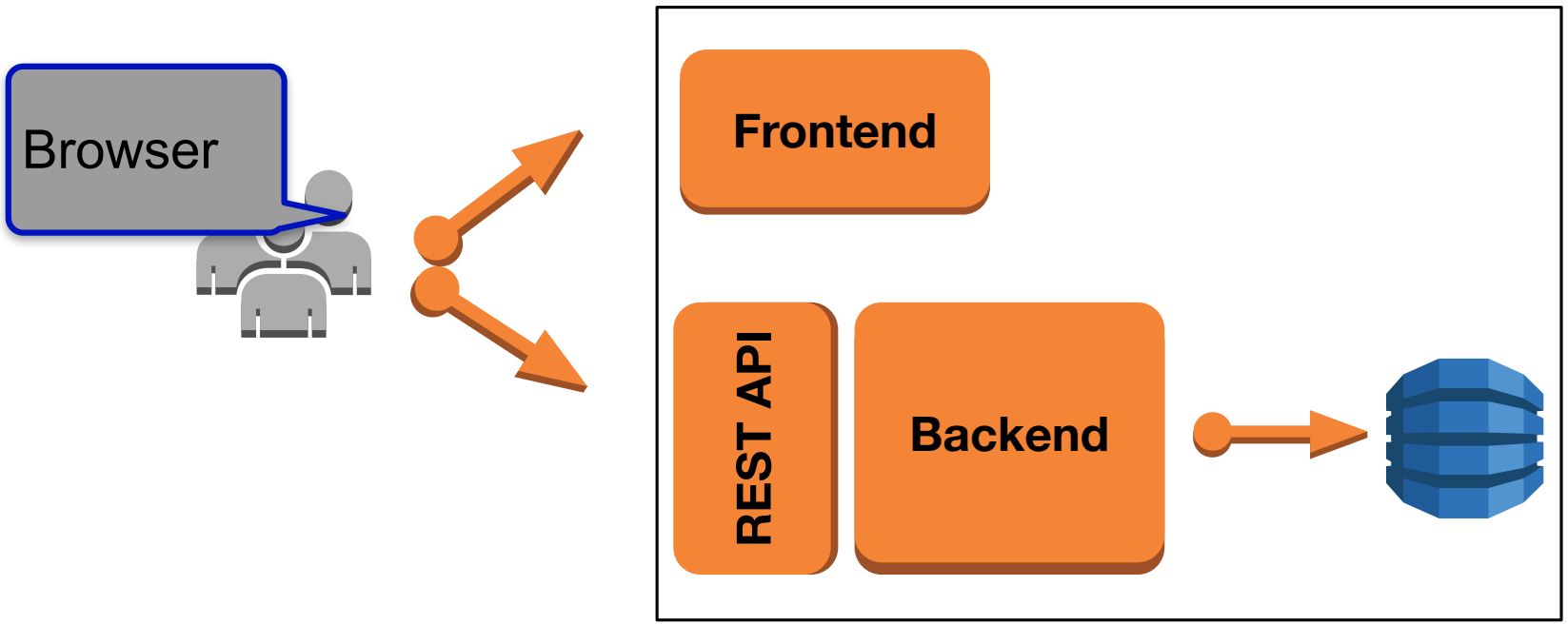
LECTURE 2

Covers …

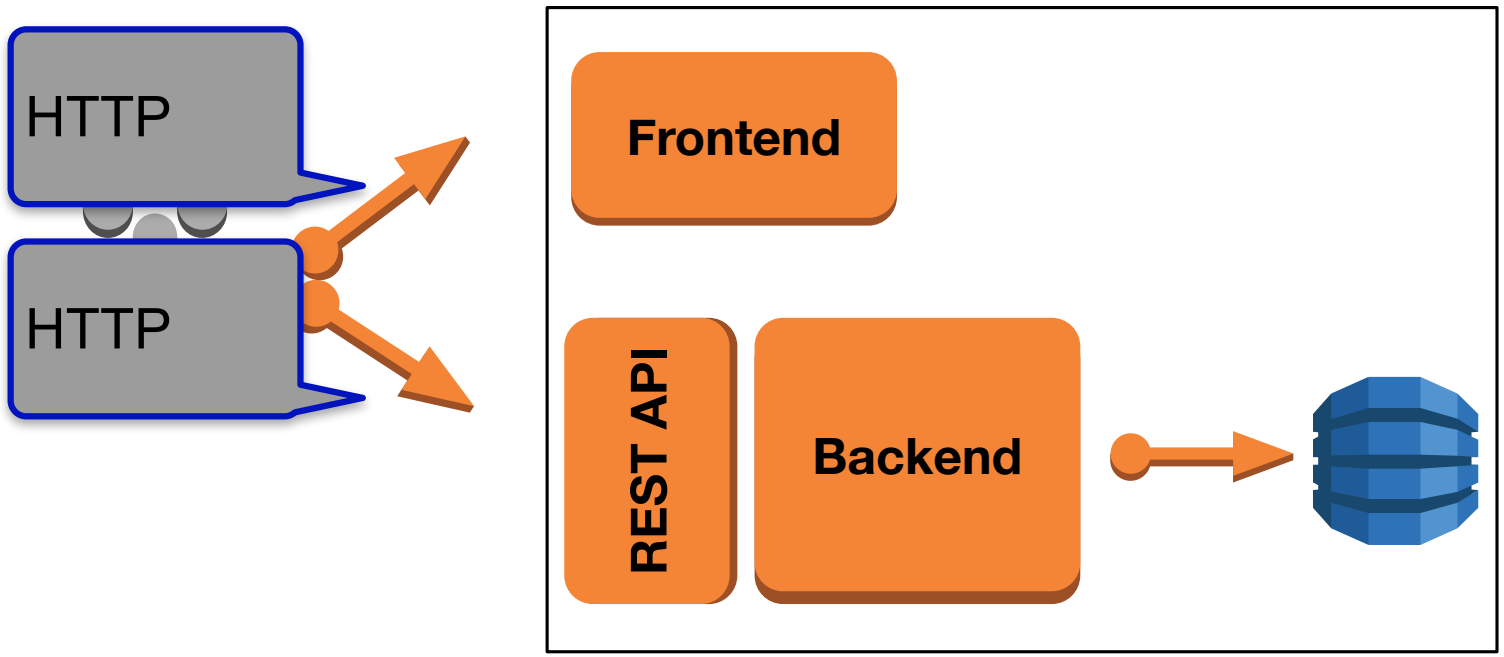
- Quick overview of fullstack Web development with MEVN
- Overview of the ScalyShop example application

ScalyShop Demo

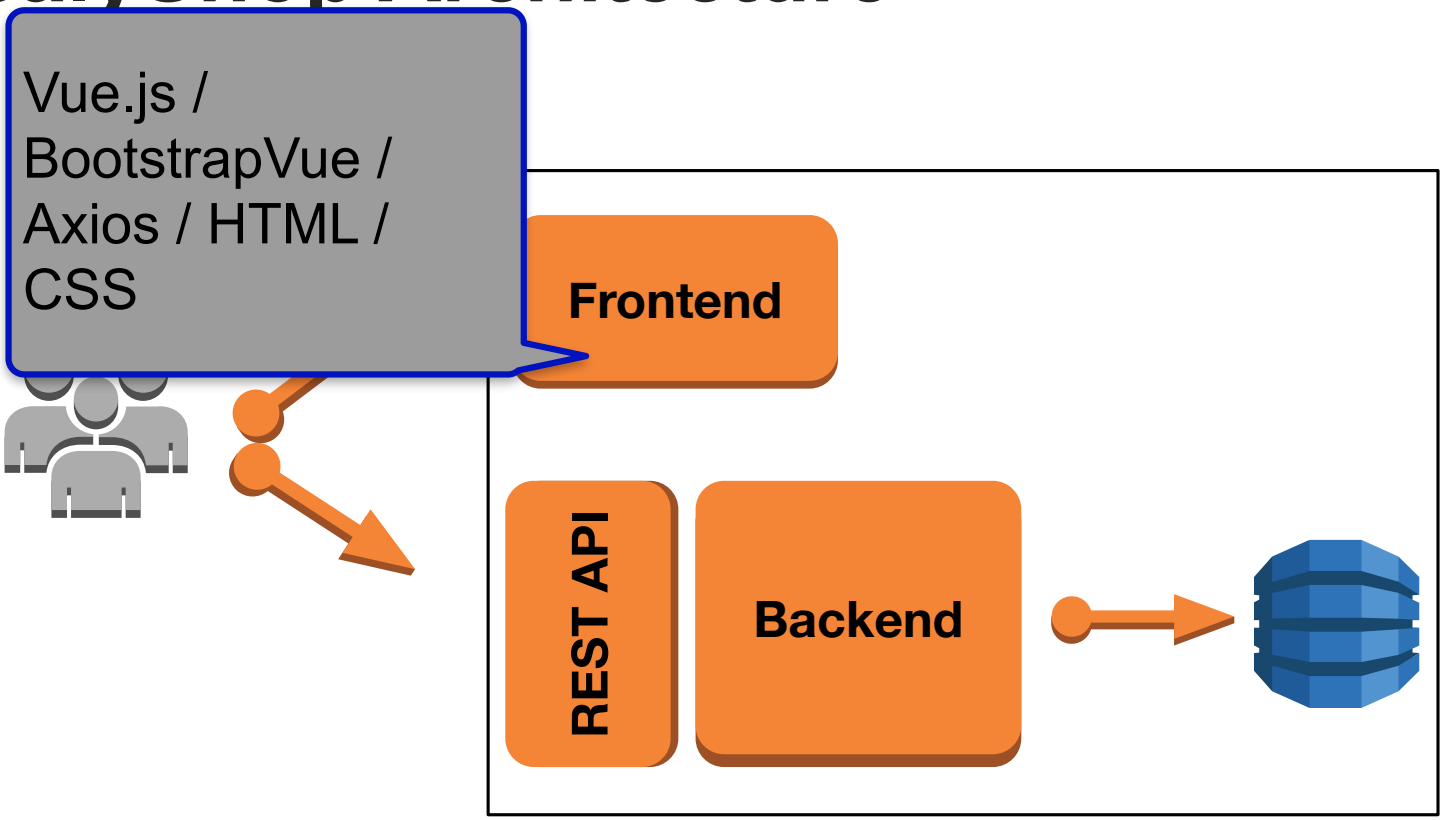
ScalyShop Architecture



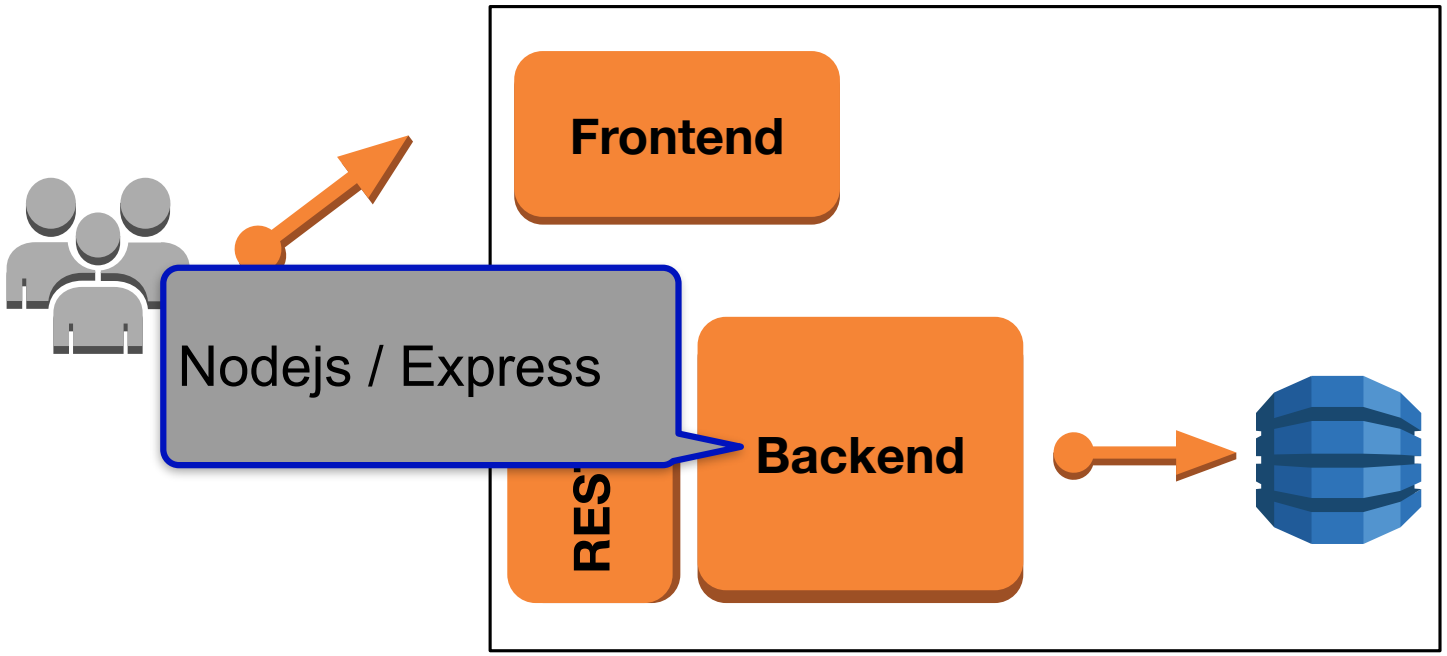
ScalyShop Architecture



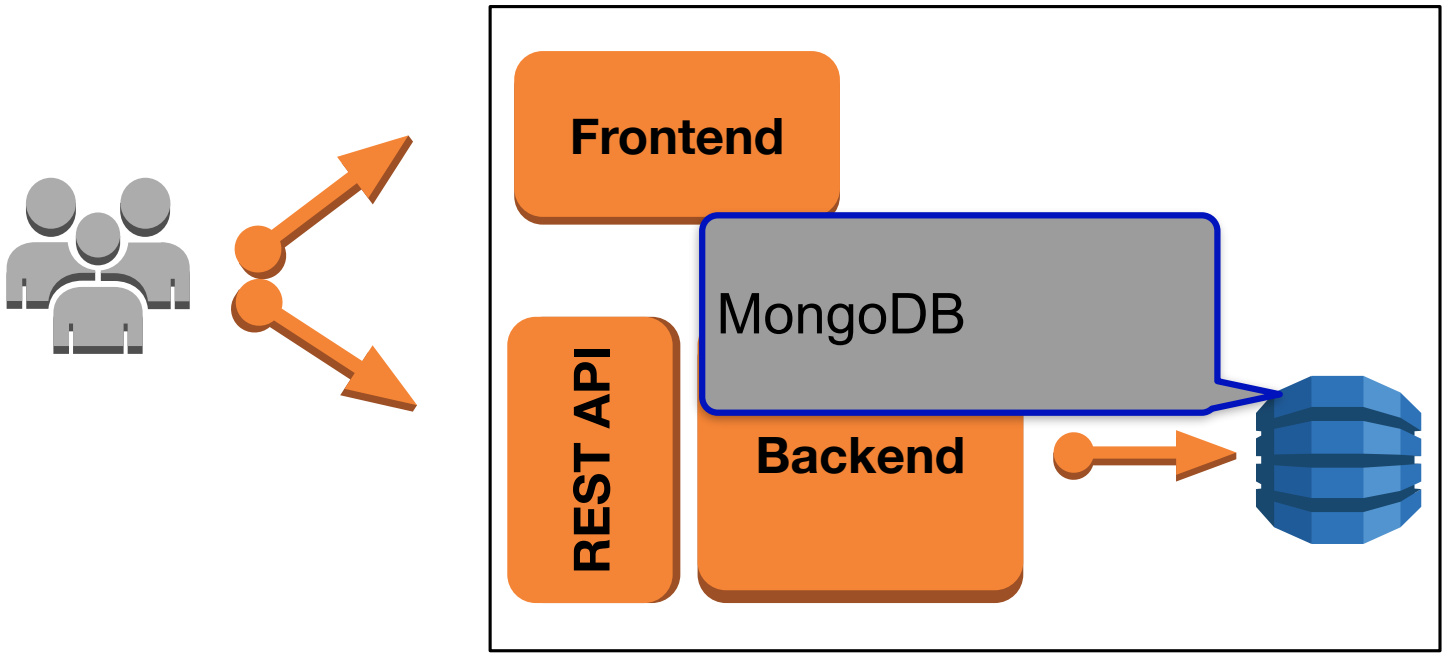
ScalyShop Architecture



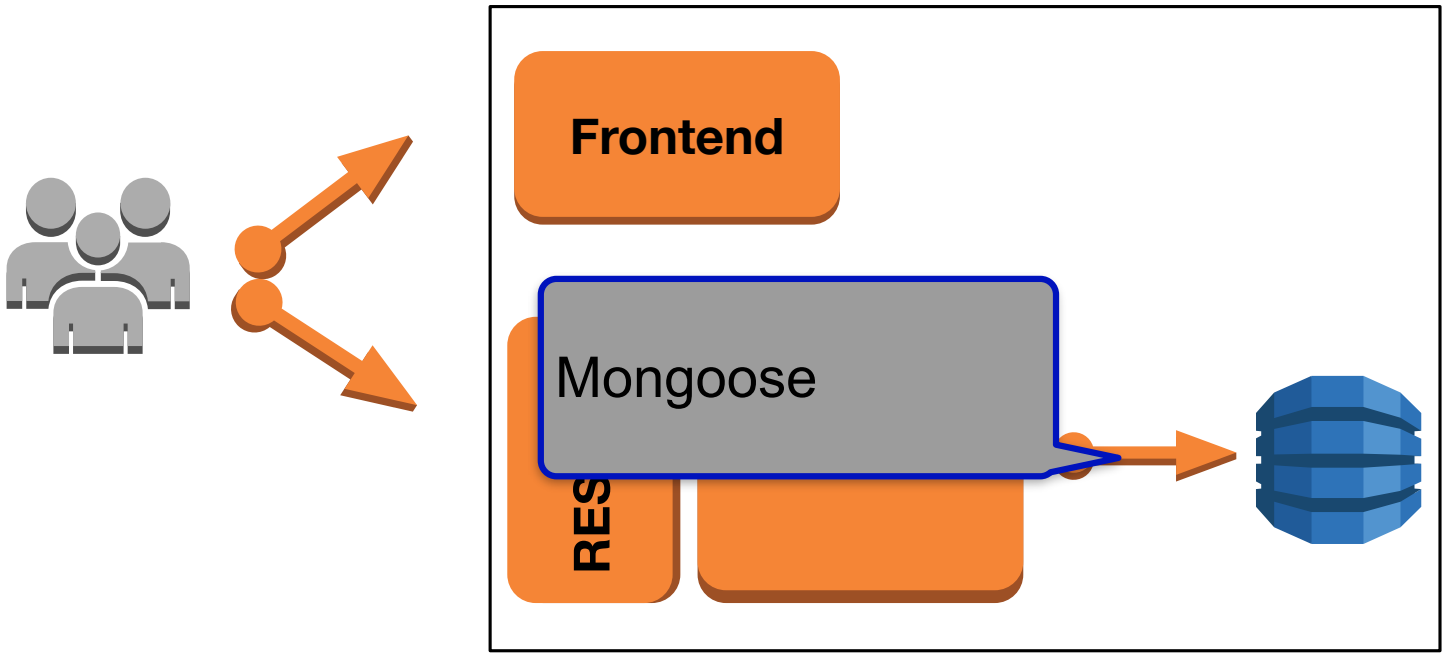
ScalyShop Architecture



ScalyShop Architecture



ScalyShop Architecture



Short "**MEVN**" for
MongoDB, **E**xpressJS, **V**ueJS, **N**odeJS

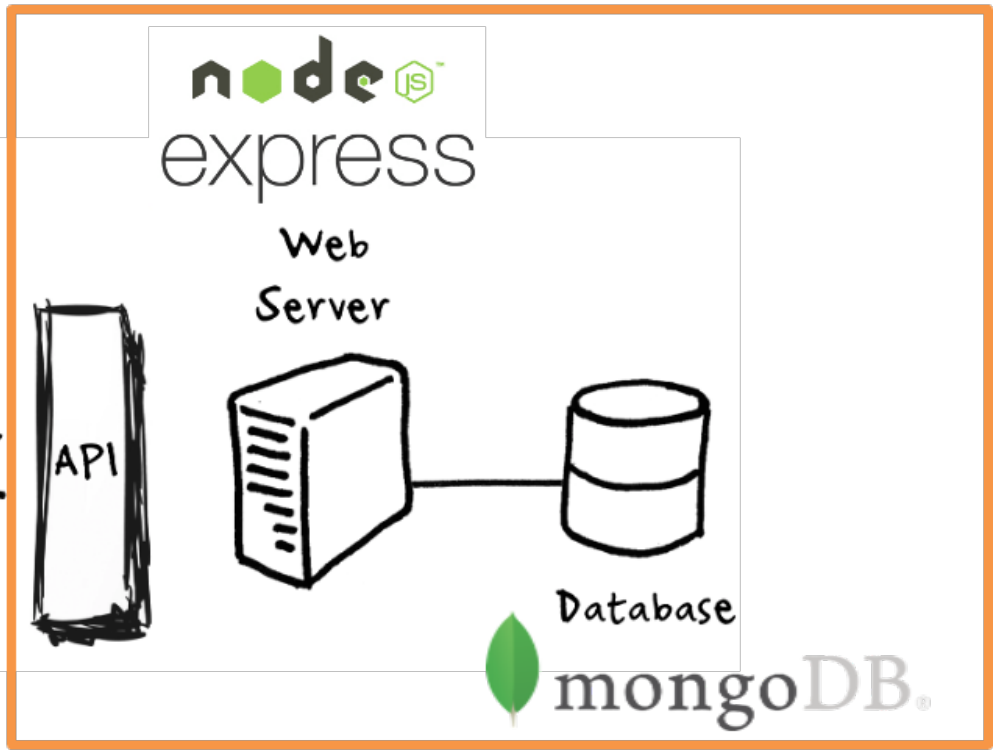
Frontend

Vue.js

BootstrapVue

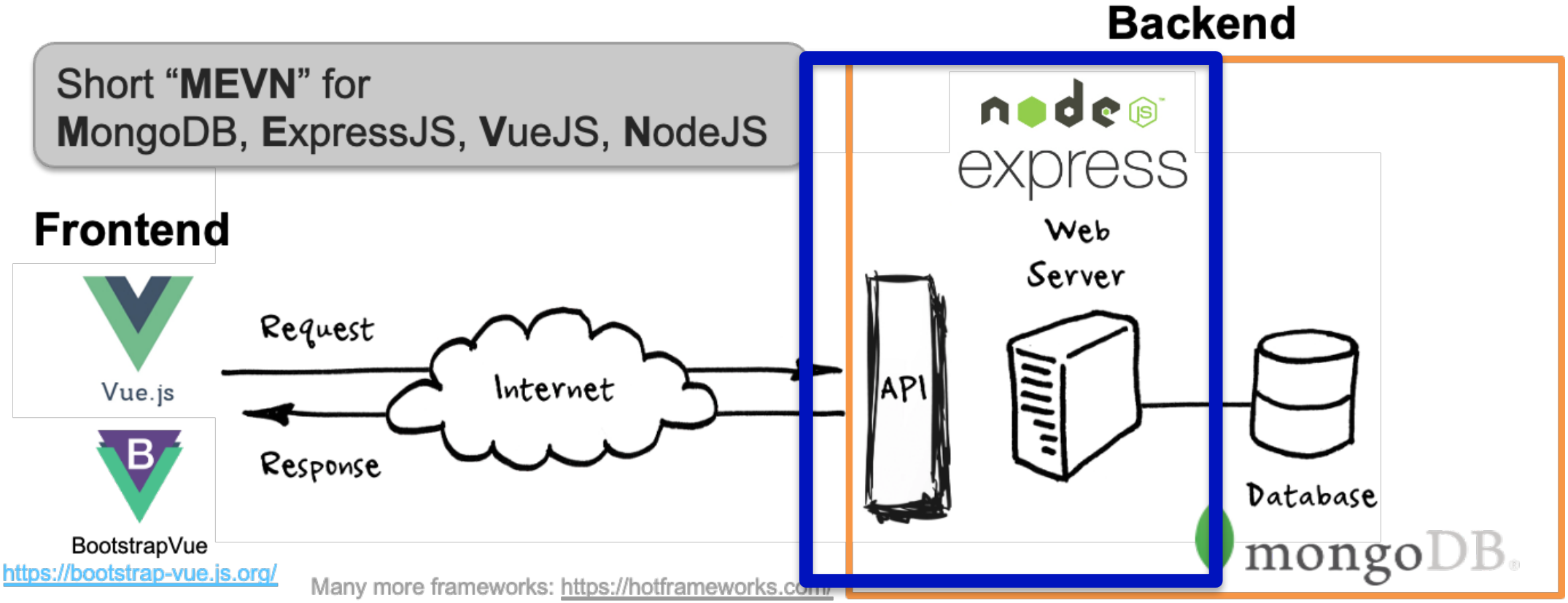
https://bootstrap-vue.js.org/

Backend



Many more frameworks: https://hotframeworks.com/

Short "**MEVN**" for
MongoDB, **E**xpressJS, **V**ueJS, **N**odeJS



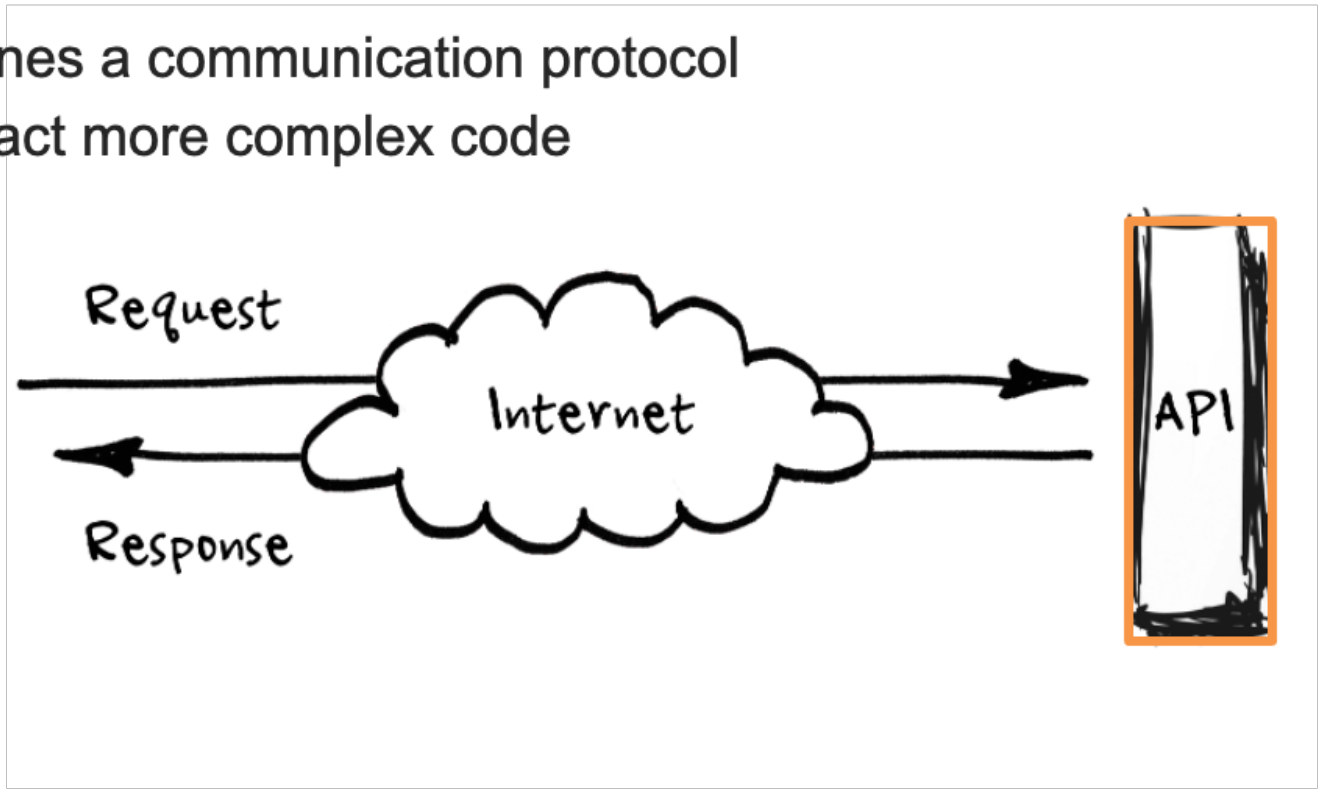
https://bootstrap-vue.js.org/

Many more frameworks: https://hotframeworks.com/

Application Programming Interface (API)

Clearly defines a communication protocol

Goal: Abstract more complex code



REST Constraints

- Client-Server
- Stateless
- Cacheable
- Layered System
- Uniform Interface
- Code-On-Demand (optional)

REST Constraints

- Client-Server
- Stateless
- Cacheable
- Layered System
- Uniform Interface
- Code-On-Demand (optional)

REST Constraints – Stateless

Client

Request

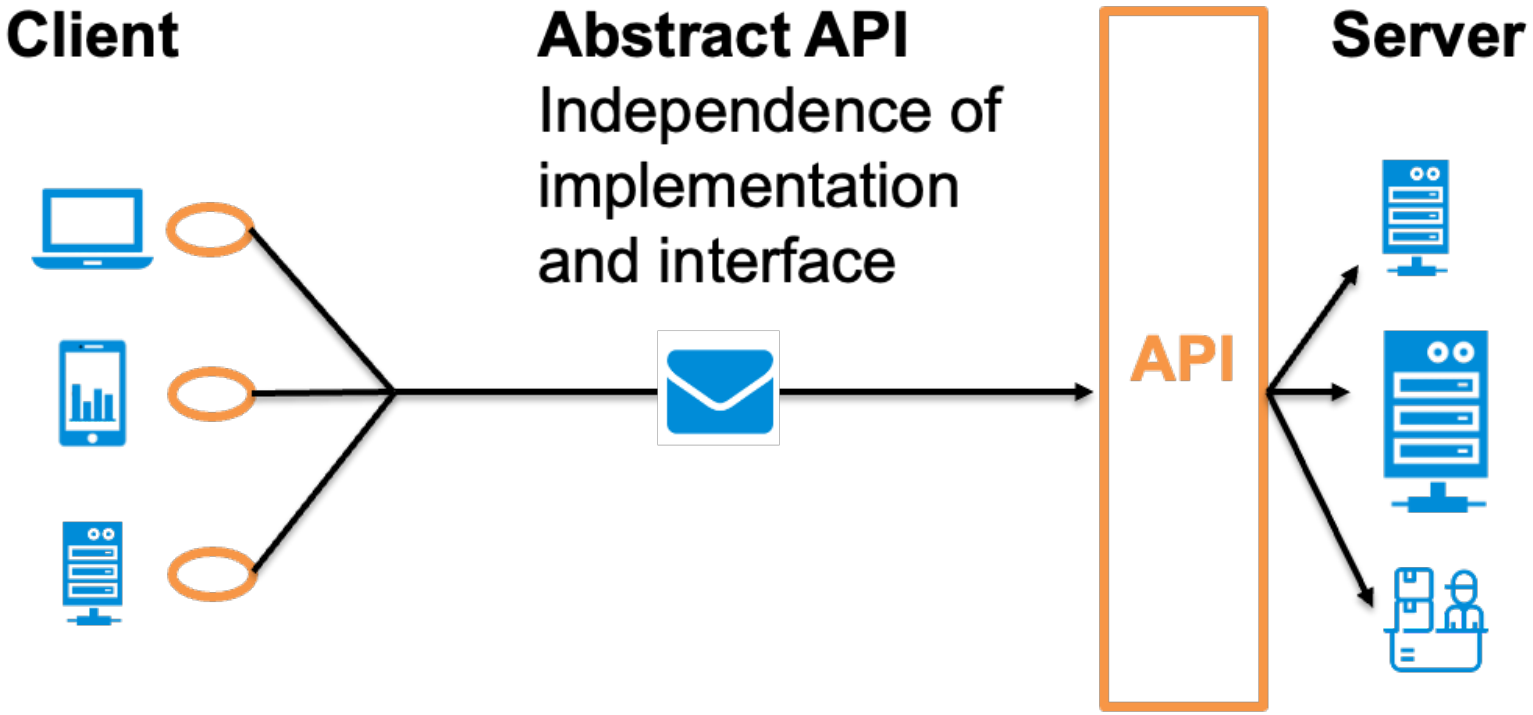
Each request must be **self-containing** (i.e., information about request and client state)

Server

Stores **no client context** (i.e., information that persists multiple requests)

REST Constraints – Uniform Interface

Key Constraint:
Distinguishes REST from other styles



RESTful API Example

Resource	POST (create)	GET (read)	PUT (update)	DELETE (delete)
`/camels`	Creates a new camel	Returns a list of camels	Bulk update all camels	Delete all camels
`/camels/2`	Method not allowed (error 405)	Returns the camel with id 2	Updates the camel with id 2	Deletes the camel with id 2

ScalyShop Backend API

API Stem:

http[s]://[servername]:[port]/api

Two entity types:

Product (/api/products, see controllers/products.js)
Order (/api/orders, see controllers/orders.js)

Example Endpoints:

GET /api/orders (fetch all orders)
DELETE /api/orders/:id (delete order by ID)
POST /api/products (add new product)
…

No authentication

Database

Backend uses a MongoDB database

- NoSQL document-oriented database system
- Easy to scale, easy to replicate

Mongoose used for Object-Relational Mapping (ORM)

- Essentially translating JS objects into MongoDB documents

① ② ③

```
// Import Mongoose
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
// Connect to MongoDB
var db = mongoose.connect(
     'mongodb://localhost:27017/animals',
     { useNewUrlParser: true });

// Define Mongoose camel schema
var camelSchema = new Schema({
     color: { type: String },
     position: { type: Number }
});
// Compile model from schema
var Camel = mongoose.model('camels', camelSchema);
```

④

```
// Create an instance of the Camel model
var myCamel = new Camel({
  color: "orange",
  position: 3
});
// Save the model
myCamel.save(function(err) {
  if (err) { return console.log(err); }
  console.log('saved!');
});
```

Code: https://git.ita.chalmers.se/courses/dit341/nodejs-intro/tree/master/mongoose-demo

Mongoose Models

Check the folder “models” for the Mongoose schema definitions

(one per REST resource)

```
var productSchema = new Schema({
    name : { type: String },
    category : { type: String },
    price : { type: Number },
    nrReserved: { type: Number },
    nrOrdered: { type: Number }
});
```

```
var orderSchema = new Schema({
    orderRef : { type: String },
    totalPrice : { type: Number },
    productsList : [
        {
        type: String,
        ref: "Product"
        }
    ],
    orderStatus : { type: String }
});
```

Endpoint Implementations

Endpoint implementations are largely straight-forward

(check the files in the “controllers” folder)

```
// Add a new product
router.post('/api/products', function(req, res, next) {
    var newproduct = new Product(req.body);
    newproduct.save(function (error) {
        if (error) {
            console.log('Error storing object: '+error);
            return res.status(400).json({'message': 'Error storing object: '+error});
        }
    });
    return res.status(201).json(newproduct);
});
```

Starting the Backend

Downloading npm dependencies:

`npm install`

Running the backend:

`npm dev` (test mode, with hot loading)

`npm start` (production mode)

Checking if the backend is online (when running locally):

Check routes “/“ or “/api/serverstatus”

Example: `wget http://localhost:5000/api/serverstatus`

Configuring the Backend

The backend (and frontend) use environment variables for configuration

Very common approach, allows you to set configuration parameters such as database hostname, ports, etc. differently when testing locally than when deploying

Sensible defaults for local testing, but you'll need to change them for different tasks in the assignments

Use these env variables, don't hardcode these values. You will run into problems later in the course if you hardcode them!

Configuring the Backend

Available environment variables:

(See also app.js, lines 13 - 18)

- PORT (default 5045)
- MONGODB_HOST (default “localhost”)
- MONGODB_DB (default “scalyDB”)
- MONGODB_PORT (default “27017”)
- MONGODB_USER (no default, will use “root”)
- MONGODB_PW (no default, equivalent to no password)

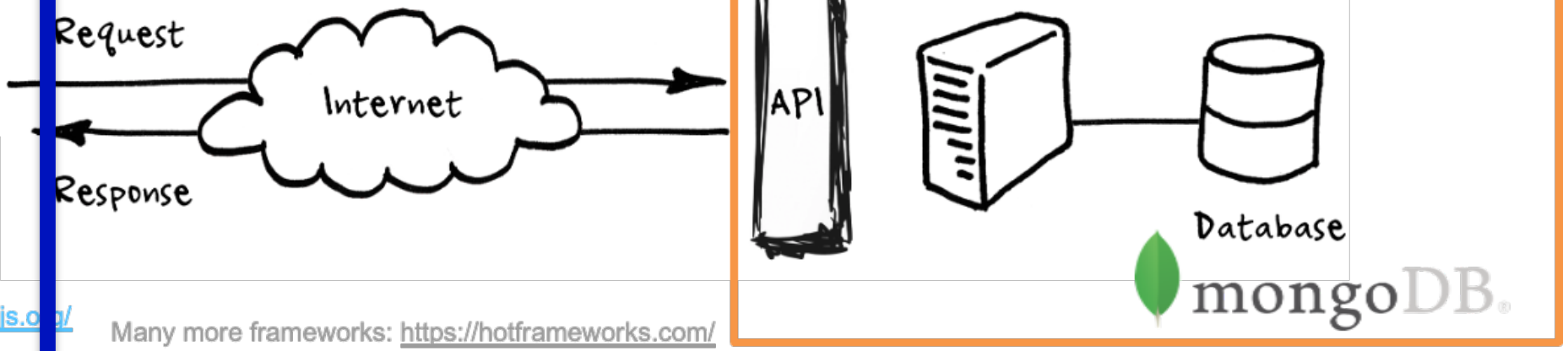
Short "**MEVN**" for
MongoDB, **E**xpressJS, **V**ueJS, **N**odeJS

Frontend

Vue.js

BootstrapVue
https://bootstrap-vue.js.org/

Backend



Many more frameworks: https://hotframeworks.com/

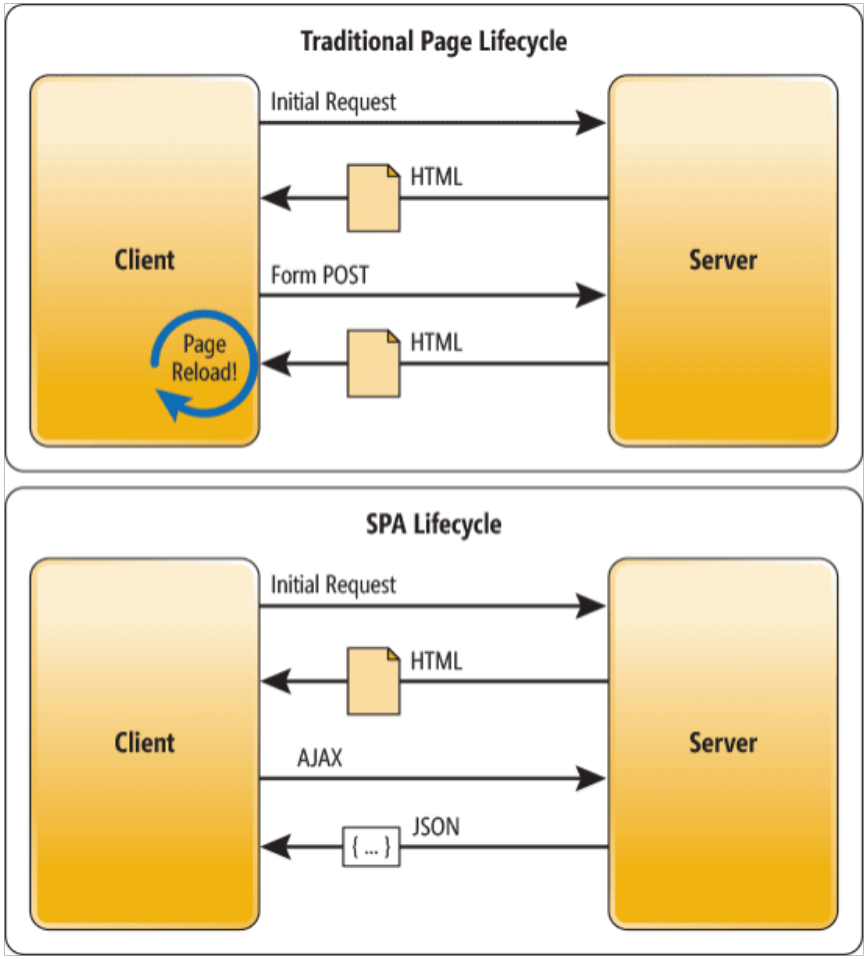
Frontend Overview

Frontend is a “rich client”

- JS code gets downloaded from frontend server upon initial request
- Browser executes frontend code
- Communicates with backend through AJAX / HTTP

Implemented using Vue.js and BootstrapVue

- Vue.js -> SPA framework (think React, but -maybe- easier)
- BootstrapVue -> UI component library


Traditional Page Lifecycle
Client
Server
Initial Request
HTML
Form POST
Page Reload!
HTML
SPA Lifecycle
Client
Server
Initial Request
HTML
AJAX
JSON
{ ... }

Frontend Overview

VueJS is reactive

A VueJS application has a **state**

View (HTML code) is generated based on the state

View is updated based on changes in the state

View can bind function calls to events

E.g., buttons being clicked

Frontend Overview

Components

Views are assembled from **components**

Two types:

- View components (represent one page in the app, folder ‘views’)
- Reusable components (building blocks that are used in different view components, folder ‘components’)

Main page:

File ‘App.vue’

VueJS Tutorial

Very good tutorial if you really want to understand VueJS:

https://www.vuemastery.com/courses/intro-to-vue-js/vue-instance/

(Exercises 1-10, the free exercises)

Starting the Frontend

Downloading npm dependencies:

`npm install`

Linting:

`npm lint` (the frontend uses *incredibly* nitpicky syntax rules, use the linter to automatically fix)

Running the frontend server:

`npm serve`

Checking if the frontend server is online (when running locally):

Open in browser, check if home page indicates that backend can be talked to

Example: open in browser `http://localhost:5046/`

Configuring the Frontend

The frontend also use environment variables for configuration

However, you need to remember that we need to set env variables **in the browser** now

Vue.js supports this through a file called ".env"

Documentation:

https://cli.vuejs.org/guide/mode-and-env.html#modes

Configuring the Frontend

Available environment variables:

(See also Api.js, lines 7 - 9)

- PORT (default “5046”)
- VUE_APP_BACKEND_HOST (default “localhost”)
- VUE_APP_BACKEND_PORT (default “5045”)
- VUE_APP_BACKEND_PROTOCOL (default “http”)

(It is important that VueJS env variables start with VUE_APP_* and go into a .env file!)