</talentlabs>

EXPRESS REVIEW & PROJECT

</talentlabs>

AGENDA

- Express Module Review
- Project 2 Overview

Review 1 - Routing

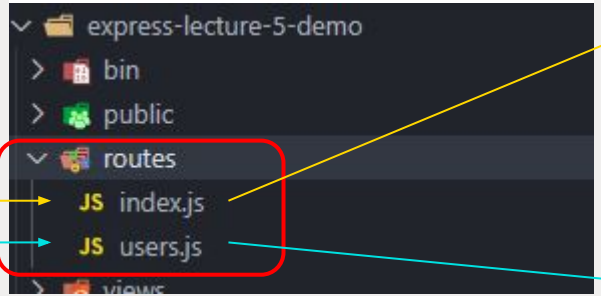
</talentlabs>

Router

1. Routers are **used** by the our Express Application in the **app.js** file.

```
app.use('/', indexRouter);
app.use('/users', usersRouter);
```

2. Each file in the **"/routes"** folder actually contains an **Express Router**. We are **building routes to a router.**



2 Routers

JS index.js × JS users.js

express > express-lecture-5-demo > routes > JS index.js > ...

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

JS index.js JS users.js ×

express > express-lecture-5-demo > routes > JS users.js > ...

```
var express = require('express');
var router = express.Router();

/* GET users listing. */
router.get('/', function(req, res, next) {
  res.send('respond with a resource');
});

module.exports = router;
```

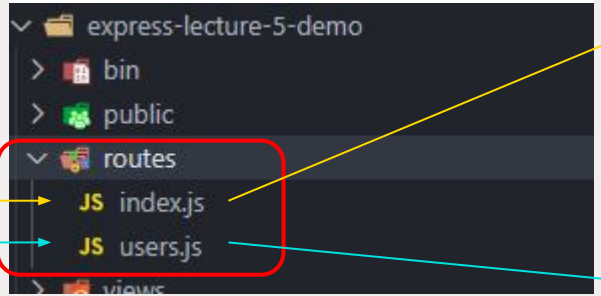

</talentlabs>

Router

1. Routers are **used** by the our Express Application in the **app.js** file.

```
app.use('/', indexRouter);
app.use('/users', usersRouter);
```

2. Each file in the **"/routes"** folder actually contains an **Express Router**. We are **building routes to a router.**



2 Routers

JS index.js × JS users.js

express > express-lecture-5-demo > routes > JS index.js > ...

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

JS index.js JS users.js ×

express > express-lecture-5-demo > routes > JS users.js > ...

```
var express = require('express');
var router = express.Router();

/* GET users listing. */
router.get('/', function(req, res, next) {
  res.send('respond with a resource');
});

module.exports = router;
```

3. We can defines **Routes** under a **Router**.

</talentlabs>

Router + Route Path

```
app.use('/', indexRouter);
app.use('/users', usersRouter);
```

When we **add a Router to the Express Application in app.js**, we can **specify a path for the Router**

JS index.js × JS users.js

express > express-lecture-5-demo > routes > JS index.js > ...

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

JS index.js JS users.js ×

express > express-lecture-5-demo > routes > JS users.js > ...

```
var express = require('express');
var router = express.Router();

/* GET users listing. */
router.get('/', function(req, res, next) {
  res.send('respond with a resource');
});

module.exports = router;
```

The resultant matching path is = Router path + Route path.

HTTP GET /

HTTP GET /users/

</talentlabs>

Review - Request Parameters

</talentlabs>

Query + Route Param

```
router.get("/demo3/:a", function (req, res, next) {
  console.log("URL Params", req.params);
  console.log("Queries", req.query);
  res.render("index", { title: "Express" });
});
```

```
http://localhost:3000/demo3/hello
> URL Params { a: 'hello' }
> Queries {}

http://localhost:3000/demo3/hello?a=10&b=20
> URL Params { a: 'hello' }
> Queries { a: '10', b: '20' }
```

Queries are optional!

Route Params are required!

</talentlabs>

Query + Route Param Data Type

```
router.get("/demo4/:a", function (req, res, next) {
  console.log("URL Params", req.params);
  console.log("Queries", req.query);
  console.log(req.query["a"] + req.query["b"])
  res.render("index", { title: "Express" });
});
```

You should expect all parameters are serialized to strings.

```
http://localhost:3000/demo4/hello?a=10&b=20
> URL Params { a: 'hello' }
> Queries { a: '10', b: '20' }
> 1020
```

They are all in string type!
'Hello', '10', '20'

'10' + '20' -> '1020' String concatenation!

We can fix this by apply the **parseInt** or the **parseFloat** function.

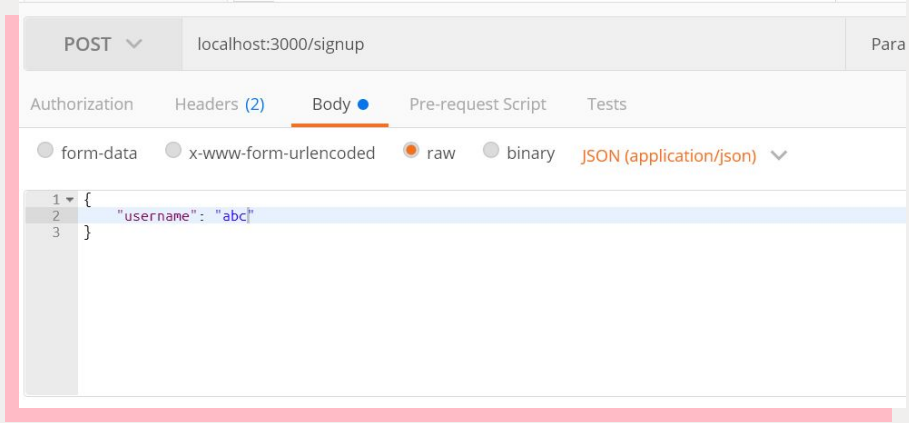
Body

```
router.post("/signup", function (req, res, next) {
  console.log("Body", req.body);
  res.render("index", { title: "Express" });
});
```

Request body are used with HTTP POST and PUT request.

It is not in the URL. We need to test this with Postman.

Body -> raw + JSON (application/json)



</talentlabs>

Review - JSON Response

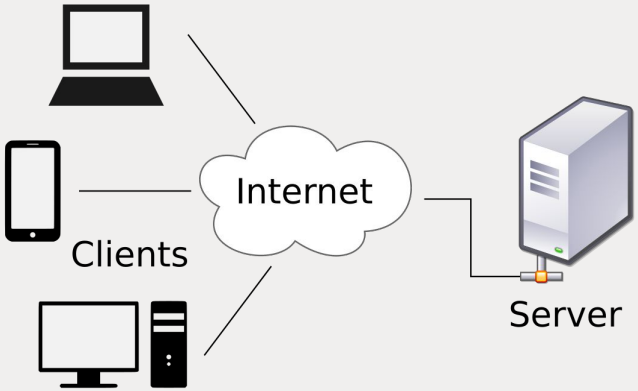
</talentlabs>

```
GET https://adventure-works.com/orders/1
```

```
{"orderId":1,"orderValue":99.90,"productId":1,"quantity":1}
```

Why JSON Response

- Not all frontend applications can read HTML.
- Same set of APIs for
  - Browser
    - React
    - Pure JavaScript
  - Mobile Application
    - Android
    - IOS
  - IoT devices
  - etc.



Can the clients read HTML?
Most of the time, only browsers read HTML.

We are going to return a JSON response.

```
// GET /customers/:id
router.get('/customers/:id', function(req, res, next) {
  console.log("Retrieve a customer with id " + req.params["id"])
  res.json({
    "id": req.params["id"],
    "name": "Anthony",
    "email": "test@test.com"
  }, 200)
});
```

The JSON

The Status code, default is 200 if it is not given.

Status: 200 OK Time: 22 ms

</talentlabs>

We are going to return a JSON response

The JSON

The Status code, default is 200 if it is not given.

```
// GET /customers/:id
router.get('/customers/:id', function(req, res, next) {
 console.log("Retrieve a customer with id " + req.params["id"])
 res.json({
  "id": req.params["id"],
  "name": "Anthony",
  "email": "test@test.com"
 }, 200)
});
```

Status: 200 OK Time: 22 ms

Review - Restful API Design

</talentlabs>

Meaningful HTTP Path and Method -> Meaningful Endpoint

A **combination of (Endpoint, HTTP method)** defines the API behaviour.

Endpoint	POST	GET	PUT	DELETE
/customers	**Bulk / Create** new customer(s)	**Retrieve** all customers	Bulk **update** of customers	**Remove** all customers
/customers/1 (/customers/:id)	**Create** a new customer with id = 1	**Retrieve** the details for customer 1	**Update** the details of customer 1 if it exists	**Remove** customer 1

Only an example, there is no 100% correct answer, the key here is to make them readable.

</talentlabs>

Meaningful HTTP Status code for response

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

200
OK

201
Created

Recall the HTTP status code in a HTTP Response. There is a list of HTTP status code. Restful API encourage us to pick a meaningful status code for a response.

</talentlabs>

Review - MySQL Database

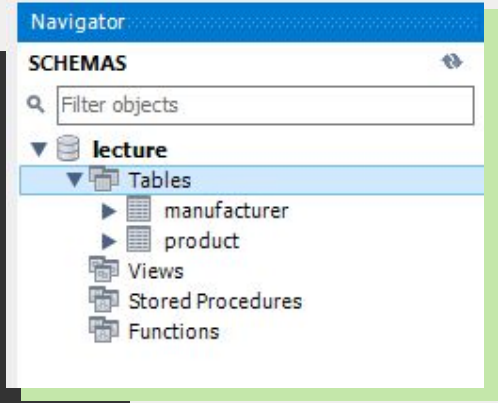
</talentlabs>

Create Table

Create a new SQL tab in the MySQL workbench and execute the following statements:

```
create table if not exists manufacturer (
      id int auto_increment primary key,
      name text
);


create table if not exists product (
    id int auto_increment primary key,
    name text,
    price decimal(19, 4),
    manufacturer_id int,
    foreign key (manufacturer_id) references manufacturer(id)
);
```



</talentlabs>

Create Table and Foreign Key Constraint

In the project, remember table creation order does matter!
If we create product table first, we will have an error.

```
create table if not exists manufacturer (
      id int auto_increment primary key,
      name text
);


create table if not exists product (
    id int auto_increment primary key,
    name text,
    price decimal(19, 4),
    manufacturer_id int,
    foreign key (manufacturer_id) references manufacturer(id)
);
```


</talentlabs>

Foreign Key Constraint

```
create table if not exists manufacturer (
    id int auto_increment primary key,
    name text
);

create table if not exists product (
    id int auto_increment primary key,
    name text,
    price decimal(19, 4),
    manufacturer_id int,
    foreign key (manufacturer_id) references manufacturer(id)
);
```

Foreign key is used to ensure the **data integrity (Referential integrity)**. A foreign key is a set of attributes in a table that **refers to the primary key of another table. The foreign key links these two tables.**

Because the database management system enforces referential constraints, it must **ensure data integrity if rows in a referenced table are to be deleted (or updated)**. If dependent rows in referencing tables still exist, those references have to be considered.

Filtering

Get the **product with id = 1:**

```
select * from product where id = 1;
```

Get the **a list products with manufacturer_id = 1:**

```
select * from product where manufacturer_id = 1;
```


</talentlabs>

Join Tables

```
select * from product
join manufacturer
 on product.manufacturer_id = manufacturer.id;
```

Get the **product with id = 1 + the details of the manufacturer:**

```
select
 product.id as id,
 product.name as name,
 product.price as price,
 product.manufacturer_id as manufacturer_id,
 manufacturer.name as manufacturer_name
from product
join manufacturer
 on product.manufacturer_id = manufacturer.id
where product.id = 1;
```

Get the **product with manufacturer_id = 1 + the details of the manufacturer:**

```
select
 product.id as id,
 product.name as name,
 product.price as price,
 product.manufacturer_id as manufacturer_id,
 manufacturer.name as manufacturer_name
from product
join manufacturer
 on product.manufacturer_id = manufacturer.id
where product.manufacturer_id = 1;
```


</talentlabs>

Review - Integrate MySQL to Express

</talentlabs>

Setup Knex - connect to the database

We will use **Knex.js** to manage our database design in Express. To install Knex we need to run:

```
npm install knex mysql --save
npx knex init
```

Next we need to tell **Knex how to connect to our database**. We need to update the database configuration in the **"knexfile.js" file**.

```
  development: {
    client: "mysql",
    connection: {
      host: "student-mysql.ccttwiegufhh.us-east-2.rds.amazonaws.com",
      user: "studentmysql",
      password: "studentmysql",
      database: "express_lecture",
    },
  },
```

Replace with your database name

</talentlabs>

What is a Seed File?

- Sometimes we want to have some **initial data**. These are called seed data.
- For example, before setting up an Admin Panel for creating Products of a online store, we can have some **initial Products inserted to the database**.

In knex, there is a concept called **seed**. A seed is a file that populates the initial data into the database.

First, Let's create 2 seed files

```
npx knex seed:make initial-manufacturer
npx knex seed:make initial-product
```

Upsert Seed Data

We want to use upsert instead.
Upsert: Update or Insert.

The upsert operation will

1. Insert a new row if there is no duplication
2. Update the existing row if there is a duplication.

./seeds/initial-manufacturer.js

```
exports.seed = function(knex) {
  return knex.raw(
    `
    insert into manufacturer (id, name)
        values (1, "Lego"), (2, "Disney")
    as new_data
    on duplicate key update
        name=new_data.name;
    `
  );
};
```

Here we define what to update when there is a duplication found.

We want to insert 2 rows,
(1, “Lego”)
(2, “Disney”)

</talentlabs>

Upsert Seed Data

We want to use upsert instead.
Upsert: Update or Insert.

The upsert operation will

1. Insert a new row if there is no duplication
2. Update the existing row if there is a duplication.

./seeds/initial-manufacturer.js

```
exports.seed = function(knex) {
  return knex.raw(
    `
    insert into product (id, name, price,
manufacturer_id)
        values (1, "Product 1", 99.9, 1), (2, "Product
2", 90.2, 2)
    as new_data
    on duplicate key update
        name=new_data.name,
        price=new_data.price,
        manufacturer_id=new_data.manufacturer_id;
    `
  );
};
```

Here we define what to update when there is a duplication found.

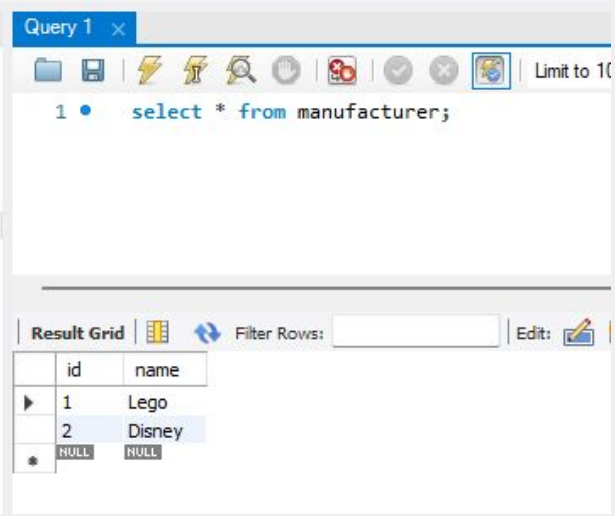
We want to insert 2 rows,
(1, "Product 1", 99.9, 1)
(2, "Product 2", 90.2, 2)

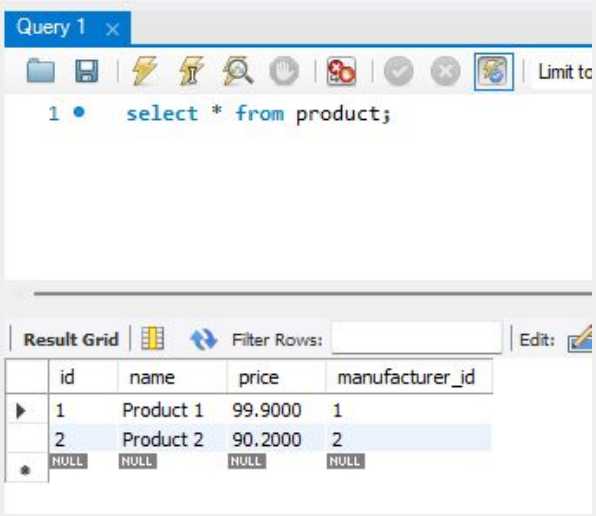
Execute the Seed File

Run these to execute the 2 seed files we created:

```
npx knex seed:run --specific=initial-manufacturer.js
npx knex seed:run --specific=initial-product.js
```

Check in MySQL Workbench





</talentlabs>

Initialize a Knex Connection

Create a file called **./database.js**:

```
const environment = process.env.NODE_ENV || "development";
// Import the knex config from the knexfile.js file.
const config = require("./knexfile");
// Pick the correct database configuration for the environment
// (such as "development")
const environmentConfig = config[environment];
const knex = require("knex");
// Create a Database Connection
const connection = knex(environmentConfig);
module.exports = connection;
```

The knexfile.js

Use the Knex Connection

./routes/index.js

```
var express = require('express');
var router = express.Router();
var connection = require('../database.js')

/* List manufacturers */
router.get('/manufacturers', function(req, res, next) {
  //knex connection
  connection
    .raw(`select * from manufacturer;`) // it is a promise
    .then(function (result) {
      var manufacturers = result[0];
      // send back the query result as json
      res.json({
        manufacturers: manufacturers,
      });
    })
    .catch(function (error) {
      // log the error
      console.log(error);
      res.json(500, {
        "message": error
      });
    });
});

module.exports = router;
```

connection.raw: run the SQL statement with the knex database connection. It returns a Promise!

If the SQL statement is executed correctly: we return the SQL results

Otherwise, we notify the client we have an Server error.

</talentlabs>

Project Overview

</talentlabs>

Task 1: Database Design
(Lecture 10)

1. Pick a data domain for your project. It must contains **at least 2 tables and 1 relationship**. We encourage you to use some real data online. For example:
   - (Manufacturer and Product) https://www.kaggle.com/PromptCloudHQ/toy-products-on-amazon
   - (Country and University) https://www.kaggle.com/mylesoneill/world-university-rankings
   - (Company and Job) https://www.kaggle.com/madhab/jobposts
   - Pick your own
2. **Document the tables' structure and their relationship**. It should include:
   - Table Name
   - Columns' name and data type (No need to include all columns from the CSV files above)
   - Relationship(s) and FK(s)
3. **Create the tables** via MySQL workbench.

Attach the documentation to the project submission report (a pdf file).

</talentlabs>

Task 1: Database Design

(Lecture 10)

Detail Compact Column 10 of 17 columns

uniq_id	product_n...	manufactu...	price	number_av...	numbe
1cde07d3e45fdfb1b1966368d3b7a522	Lego City 2824: Advent Calendar 2010	LEGO	£43.87	19 new	40
900e782753c3cc6a6396da3315add573	LEGO Friends 41016: Advent Calendar	LEGO	£24.95	19 new	87
15e03b806b3d90b	LEGO Star Wars	LEGO	£68.87	16 new	30

Pick some interesting columns from the dataset and model them correctly.

```
create table if not exists manufacturer (
      id int auto_increment primary key,
      name text
);


create table if not exists product (
    id int auto_increment primary key,
    name text,
    price decimal(19, 4),
    manufacturer_id int,
    foreign key (manufacturer_id) references manufacturer(id)
);
```


</talentlabs>

Task 2: Generate Seed Data
(Lecture 11)

1. Setup the knex connection configuration.
2. Create 1 (or more) seed files for each tables.
3. Convert the CSV data into INSERT SQL statements for the seed file. Please **at least insert 20 rows** of seed data.
4. Execute the seed files.

Attach a screenshot showing the inserted data via MySQL workbench to the project submission report (a pdf file).

</talentlabs>

Task 2: Generate Seed Data

(Lecture 11)

Limit to 1000 rows

```
1   select * from product
```

Result Grid | Filter Rows: | Edit: | Export/Import:

	id	name	price	manufacturer_id
▶	1	Lego City 2824: Advent Calendar 2010	43.8700	1
	2	LEGO Friends 41016: Advent Calendar	24.9500	1
	3	LEGO Star Wars 75018: Jek-14's Stealth Starfig...	68.8700	1
	4	LEGO City 7687 Advent Calendar 2009	24.9900	1
	5	LEGO Star Wars 75015: Corporate Alliance Tan...	39.9900	1
	6	LEGO City 7907: Advent Calendar	61.1400	1
	7	Lego Kingdoms Advent Calendar 7952r	69.9900	1
	8	Lego Castle Advent Calendar 7979 NEW / SEALED	65.4800	1
	9	DESPICABLE ME 2 - Minion cuddly Soft Toy - Plu...	19.9900	2
	10	Disney Mickey 2-in-1 Cushion Plush	8.0500	2
	11	Lanyard With Dopey Head Dangle - Snow White...	9.7100	2
	12	John Adams Disney Frozen My Blopens Set	5.4800	2
	13	Disney Cars 2 Die Cast 1:55 Scale Mel Dorado #27	9.6900	2
	14	Disney Pixar Cars Diecast Boost With Flames	14.9900	2
	15	Anker Princess Play Pack	2.3100	2
	16	Hornby 2014 Catalogue	3.4200	3
	17	HORNBY Coach R4410A BR Hawksworth Corrid...	39.9900	3
	18	Hornby 00 Gauge 0-4-0 Gildenlow Salt Co. Stea...	32.1900	3
	19	Hornby 00 Gauge 230mm BR Bogie Passenger B...	24.9900	3
	20	Hornby Santa's Express Train Set	69.9300	3
*	NULL	NULL	NULL	NULL

Limit to 1000 rows

```
2   p.id, p.name, p.price, m.name as manufacturer
3   from product as p
4   join manufacturer as m
5   on p.manufacturer_id = m.id;
6
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	id	name	price	manufacturer
▶	1	Lego City 2824: Advent Calendar 2010	43.8700	LEGO
	2	LEGO Friends 41016: Advent Calendar	24.9500	LEGO
	3	LEGO Star Wars 75018: Jek-14's Stealth Starfig...	68.8700	LEGO
	4	LEGO City 7687 Advent Calendar 2009	24.9900	LEGO
	5	LEGO Star Wars 75015: Corporate Alliance Tan...	39.9900	LEGO
	6	LEGO City 7907: Advent Calendar	61.1400	LEGO
	7	Lego Kingdoms Advent Calendar 7952r	69.9900	LEGO
	8	Lego Castle Advent Calendar 7979 NEW / SEALED	65.4800	LEGO
	9	DESPICABLE ME 2 - Minion cuddly Soft Toy - Plu...	19.9900	Disney
	10	Disney Mickey 2-in-1 Cushion Plush	8.0500	Disney
	11	Lanyard With Dopey Head Dangle - Snow White...	9.7100	Disney
	12	John Adams Disney Frozen My Blopens Set	5.4800	Disney
	13	Disney Cars 2 Die Cast 1:55 Scale Mel Dorado #27	9.6900	Disney
	14	Disney Pixar Cars Diecast Boost With Flames	14.9900	Disney
	15	Anker Princess Play Pack	2.3100	Disney
	16	Hornby 2014 Catalogue	3.4200	Hornby
	17	HORNBY Coach R4410A BR Hawksworth Corrid...	39.9900	Hornby
	18	Hornby 00 Gauge 0-4-0 Gildenlow Salt Co. Stea...	32.1900	Hornby
	19	Hornby 00 Gauge 230mm BR Bogie Passenger B...	24.9900	Hornby
	20	Hornby Santa's Express Train Set	69.9300	Hornby

</talentlabs>

Task 3: Restful API Design
(Lecture 7)

1. Review the following real world API documentation:
   - https://stripe.com/docs/api/customers
2. **Document** the API endpoint for "**List, Retrieve, Create, Update, Delete**" **of each resource** (for example Manufacturer and Product). Each endpoint documentation should contain:
   - HTTP method
   - HTTP Path
   - Request parameters (route param or query or body)
   - Response structure
   - Status code (including failure)
   - An request-response example for the happy case
   - **At least 1 request-response example for the error case**

Attach the documentation to the project submission report (a pdf file).

</talentlabs>

Task 3: Restful API Design
(Lecture 7)

Text Description to the endpoint.

List all customers

Returns a list of your customers. The customers are returned sorted by creation date, with the most recent customers appearing first.

Parameters

email optional
A case-sensitive filter on the list based on the customer's `email` field. The value must be a string.

Request example.

GET /v1/customers Select library

```
curl https://api.stripe.com/v1/customers \
  -u sk_test_4eC39HqLyjWDarjtT1zdp7dc: \
  -d limit=3 \
  -G
```

Response example.

RESPONSE


```
{
  "object": "list",
  "url": "/v1/customers",
  "has_more": false,
  "data": [
    {
      "id": "cus_8epDebVEl8Bs2V",
      "object": "customer",
      "address": null,
      "balance": 0,
      "created": 1466202923,
      "currency": "usd",
      "default_source": "card_18NVYR2eZvKYlo2CQ2ieV9S5",
      "delinquent": true,
      "description": "Mia Wilson",
      "discount": null,
      "email": "mia.wilson.99@example.com",
      "invoice_prefix": "D86E170",
      "invoice_settings": {
        "custom_fields": null,
        "default_payment_method": null,
        "footer": null
      },
```


</talentlabs>

Task 4: Define the Express Routes
(Lecture 8, 9)

1. In routes/index.js defines the routes for the **"List, Retrieve, Create, Update, Delete" endpoint of each resource.**
    - For example if you created 2 tables, you should implement 2 * 5 = 10 routes.
2. You don't need to connect to the database at this moment. Try to return some fake data in the response.
3. **Add at least 1 parameter validation** (like not empty).
4. Test your APIs with Postman to make sure all the new endpoints are reachable.
5. (Bonus task) Try to split the routes into different router files, because there are too many routes in the index.js router file.

</talentlabs>

Task 5: Integrate MySQL
(Lecture 11)

1. Initialize a knex connection and import it to each router files you are working on.
2. Implement each of the routes with the actual SQL query (with knex).
    - It will be easier to implement in this order:
        i. List
        ii. Retrieve
        iii. Delete
        iv. Create
        v. Update

</talentlabs>

Task 5: Integrate MySQL
(Lecture 11)

GET /products

GET | localhost:3000/products/ | Params | Send | Save

Authorization | Headers (2) | Body | Pre-request Script | Tests | Code

Type: No Auth

Body | Cookies | Headers (7) | Test Results | Status: 200 OK | Time: 1102 ms

Pretty | Raw | Preview | JSON


```
{
    "products": [
        {
            "id": 1,
            "name": "Lego City 2824: Advent Calendar 2010",
            "price": 43.87,
            "manufacturer_id": 1
        },
        {
            "id": 2,
            "name": "LEGO Friends 41016: Advent Calendar",
            "price": 24.95,
            "manufacturer_id": 1
        },
        {
            "id": 3,
            "name": "LEGO Star Wars 75018: Jek-14's Stealth Starfighter",
            "price": 68.87,
            "manufacturer_id": 1
        },
        {
            "id": 4,
            "name": "LEGO City 7687 Advent Calendar 2009",
            "price": 24.99,
            "manufacturer_id": 1
        },
        {
            "id": 5,
            "name": "LEGO Star Wars 75015: Corporate Alliance Tank Droid",
            "price": 39.99,
            "manufacturer_id": 1
        },
        {
            "id": 6,
            "name": "LEGO City 7907: Advent Calendar",
```


GET /manufacturers

localhost:3000/manuf | No Env

GET | localhost:3000/manufacturers/ | Param

Authorization | Headers (2) | Body | Pre-request Script | Tests

Type: No Auth

Body | Cookies | Headers (7) | Test Results

Pretty | Raw | Preview | JSON


```
{
    "manufacturers": [
        {
            "id": 1,
            "name": "LEGO"
        },
        {
            "id": 2,
            "name": "Disney"
        },
        {
            "id": 3,
            "name": "Hornby"
        }
    ]
}
```


</talentlabs>

Task 5: Integrate MySQL
(Lecture 11)

POST /products

localhost:3000/produ

POST localhost:3000/products/

Authorization Headers (2) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   "name": "Test product",
3   "price": "2.19",
4   "manufacturer_id": "1"
5 }
```

Result Grid Filter Rows: Export: Wrap Cell Content:

id	name	price	manufacturer
1	Lego City 2824: Advent Calendar 2010	43.8700	LEGO
3	LEGO Star Wars 75018: Jek-14's Stealth Starfig...	68.8700	LEGO
4	LEGO City 7687 Advent Calendar 2009	24.9900	LEGO
5	LEGO Star Wars 75015: Corporate Alliance Tan...	39.9900	LEGO
6	LEGO City 7907: Advent Calendar	61.1400	LEGO
7	Lego Kingdoms Advent Calendar 7952r	69.9900	LEGO
8	Lego Castle Advent Calendar 7979 NEW / SEALED	65.4800	LEGO
21	Test product	2.1900	LEGO

</talentlabs>

Task 5: Integrate MySQL
(Lecture 11)

PUT /products

localhost:3000/produ

PUT localhost:3000/products/21

Authorization Headers (2) Body Pre-request

form-data x-www-form-urlencoded raw

```
1 {
2   "name": "Test product updted",
3   "price": "2.20",
4   "manufacturer_id": "2"
5 }
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	name	price	manufacturer
1	Lego City 2824: Advent Calendar 2010	43.8700	LEGO
3	LEGO Star Wars 75018: Jek-14's Stealth Starfig...	68.8700	LEGO
4	LEGO City 7687 Advent Calendar 2009	24.9900	LEGO
5	LEGO Star Wars 75015: Corporate Alliance Tan...	39.9900	LEGO
6	LEGO City 7907: Advent Calendar	61.1400	LEGO
7	Lego Kingdoms Advent Calendar 7952r	69.9900	LEGO
8	Lego Castle Advent Calendar 7979 NEW / SEALED	65.4800	LEGO
21	Test product	2.1900	LEGO

13	Disney Cars 2 Die Cast 1:55 Scale Mel Dorado #27	9.6900	Disney
14	Disney Pixar Cars Diecast Boost With Flames	14.9900	Disney
15	Anker Princess Play Pack	2.3100	Disney
21	Test product updted	2.2000	Disney

</talentlabs>

Task 6: Security
(Lecture 12, 13)

1. Make sure the Express application can **handle SQL injection** by using **Parameter Binding.**
    - Test with Postman if you can hack your own application with SQL injection.
2. Make sure the Express application can **remove dangerous JS code** in a request by using a **XSS filter**.
    - Test with Postman if you can hack your own application with XSS.

Attach the test result to the project submission report (a pdf file).

</talentlabs>

Task 6: Security
(Lecture 12, 13)

Migrating SQL Injection

localhost:3000/manuf

GET localhost:3000/manufacturers/2or1;

Authorization Headers (2) Body Pre-request Script Te

Type No Auth

Body Cookies Headers (7) Test Results

Pretty Raw Preview JSON

```
1 {
2     "manufacturer": {
3         "id": 2,
4         "name": "Disney"
5     }
6 }
```

Migrating XSS

localhost:3000/produ

PUT localhost:3000/products/21

Authorization Headers (2) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON

```
1 {
2  "name": "Test product updted <script>alert(\"HI\")</script>",
3  "price": "2.20",
4  "manufacturer_id": "2"
5 }
```

21	Test product updted	2.2000	Disney

</talentlabs>