# Project: GrubDash

You've been hired as a back-end developer for a new startup called GrubDash!  
As another developer works on the design and front-end experience, you have been tasked with setting up an API and building out specific routes so that  
the front-end developers demo out some initial design ideas to the big bosses.

This project will test your ability to build API's with complex validation.  
To succeed at this project, you'll need to demonstrate that you can do the following:

* Run tests from the command line.
* Use common middleware packages.
* Receive requests through routes.
* Access relevant information through route parameters.
* Build an API following RESTful design principles.
* Write custom middleware functions.

You will not need to make any edits to HTML or CSS for this project.

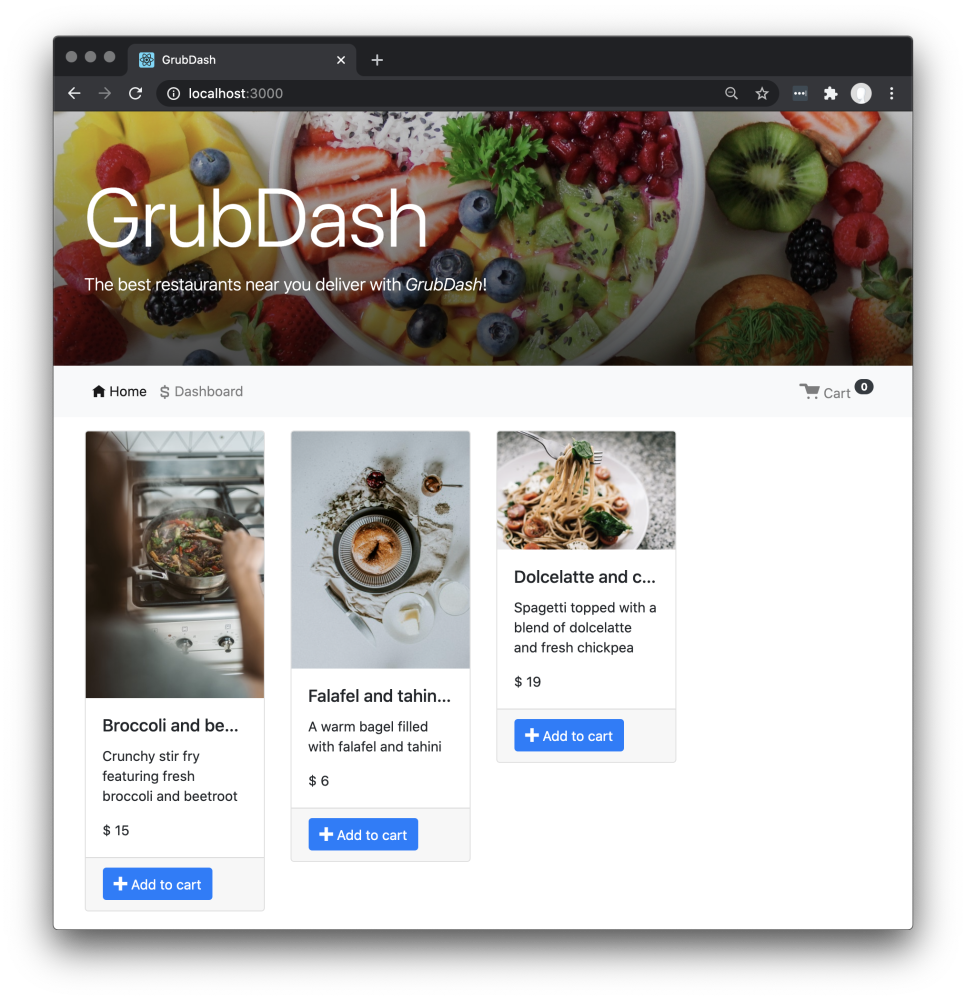
## GrubDash Front End

While it is not required, if you would like to see this project connected to a Front End Application, visit the following repository:

* [Starter Code: GrubDash Front End](https://github.com/Thinkful-Ed/starter-grub-dash-front-end)

Instructions on how to get the Front End application up and running are included in the repository.

1:1



Again, it is not necessary to connect the Front End to successfully complete this project.

## Project setup

Follow the instructions below to get this project up and running on your own machine:

* Sync this Qualified assignment with your computer.
* If you choose to work on this project disconnected from Qualified, run npm install.

## Running tests

To run the tests, you can run the following command:

npm test

## Instructions

Your goal for this checkpoint is to get the tests to pass.  
To do so, you will be creating a server to access two resources, dishes and orders in addition to error handling.  
Your server should follow the structure you've learned in the course. Complete the following tasks to pass the tests and this assignment.

### Existing files

| **file path** | **description** |
| --- | --- |
| src/app.js | Contains the Express application. You will not need to make changes to this file. |
| src/data/dishes-data.js | Contains the dishes data. This is the shape of the dish data that the API will send and receive. You can add or remove dishes if you like. |
| src/data/orders-data.js | Contains the orders data. This is the shape of the order data that the API will send and receive. You can add or remove orders if you like. |
| src/dishes/dishes.controller.js | Add middleware and handlers for dishes to this file, then export the functions for use by the router. |
| src/dishes/dishes.router.js | Add routes and attach handlers to the router exported by this file. |
| src/errors/errorHandler.js | Exports the error handler function for use by the Express application. |
| src/errors/methodNotAllowed.js | Exports the 405 Method Not Allowed handler function for use by the Express application. |
| src/errors/notFound.js | Exports the 404 Not Found handler function for use by the Express application. |
| src/orders/orders.controller.js | Add middleware and handlers for orders to this file, then export the functions for use by the router. |
| src/orders/orders.router.js | Add routes and attach handlers to the router exported by this file. |
| src/server.js | Contains the server code. You will not need to make changes to this file. |
| src/utils/nextId.js | Exports the nextId function. Use this function anytime you need to assign a new id. You will not need to make changes to this file. |
| test/dishes-router.test.js | Tests for the dishes router. You will not need to make changes to this file. |
| test/order-router.test.js | Tests for the orders router. You will not need to make changes to this file. |
| test/make-test-app.js | Function used by the tests. You will not need to make changes to this file. |

### Tasks

1. In the src/dishes/dishes.controller.js file, add handlers and middleware functions to create, read, update, and list dishes. Note that dishes cannot be deleted.
2. In the src/dishes/dishes.router.js file, add two routes: /dishes, and /dishes/:dishId and attach the handlers (create, read, update, and list) exported from src/dishes/dishes.controller.js.
3. In the src/orders/orders.controller.js file, add handlers and middleware functions to create, read, update, delete, and list orders.
4. In the src/orders/orders.router.js file, add two routes: /orders, and /orders/:orderId and attach the handlers (create, read, update, delete, and list) exported from src/orders/orders.controller.js.
5. Anytime you need to assign a new id to an order or dish, use the nextId function exported from src/utils/nextId.js

### Routes

To complete this project, you will create the following routes:

#### GET /dishes

This route will respond with a list of all existing dish data.

**Example Request**

GET http://localhost:5000/dishes

**Example Response**

{

"data": [

{

"id": "d351db2b49b69679504652ea1cf38241",

"name": "Dolcelatte and chickpea spaghetti",

"description": "Spaghetti topped with a blend of dolcelatte and fresh chickpeas",

"price": 19,

"image\_url": "https://images.pexels.com/photos/1279330/pexels-photo-1279330.jpeg?h=530&w=350"

}

]

}

#### POST /dishes

This route will save the dish and respond with the newly created dish.

##### Validation:

If any of the following validations fail, respond with a status code of 400 and an error message.

| **Validation** | **Error message** |
| --- | --- |
| name property is missing | Dish must include a name |
| name property is empty "" | Dish must include a name |
| description property is missing | Dish must include a description |
| description property is empty "" | Dish must include a description |
| price property is missing | Dish must include a price |
| price property 0 or less | Dish must have a price that is an integer greater than 0 |
| price property is not an integer | Dish must have a price that is an integer greater than 0 |
| image\_url property is missing | Dish must include a image\_url |
| image\_url property is empty "" | Dish must include a image\_url |

**Example Request**

POST http://localhost:5000/dishes

Body:

{

"data": {

"name": "Dolcelatte and chickpea spaghetti",

"description": "Spaghetti topped with a blend of dolcelatte and fresh chickpeas",

"price": 19,

"image\_url": "https://images.pexels.com/photos/1279330/pexels-photo-1279330.jpeg?h=530&w=350"

}

}

**Example Response**

Status 201

{

"data": {

"id": "d351db2b49b69679504652ea1cf38241",

"name": "Dolcelatte and chickpea spaghetti",

"description": "Spaghetti topped with a blend of dolcelatte and fresh chickpeas",

"price": 19,

"image\_url": "https://images.pexels.com/photos/1279330/pexels-photo-1279330.jpeg?h=530&w=350"

}

}

#### GET /dishes/:dishId

This route will respond with the dish where id === :dishId, or return 404 if no matching dish is found.

**Example Request**

GET http://localhost:5000/dishes/3c637d011d844ebab1205fef8a7e36ea

**Example Response**

Status 200

{

"data": {

"id": "d351db2b49b69679504652ea1cf38241",

"name": "Dolcelatte and chickpea spaghetti",

"description": "Spaghetti topped with a blend of dolcelatte and fresh chickpeas",

"price": 19,

"image\_url": "https://images.pexels.com/photos/1279330/pexels-photo-1279330.jpeg?h=530&w=350"

}

}

#### PUT /dishes/:dishId

This route will update the dish where id === :dishId, or return 404 if no matching dish is found.

##### Validation:

The update validation must include all of the same validation as the POST /dishes route, plus the following:

| **Validation** | **Error message** |
| --- | --- |
| :dishId does not exist | Dish does not exist: $dishId}. |
| id in the body does not match :dishId in the route | Dish id does not match route id. Dish: ${id}, Route: ${dishId} |

**Note**: The id property is not required in the body of the request, but if it is present it must match :dishId from the route.

**Example Request**

PUT http://localhost:5000/dishes/3c637d011d844ebab1205fef8a7e36ea

Body:

{

"data": {

"id": "3c637d011d844ebab1205fef8a7e36ea",

"name": "Century Eggs",

"description": "Whole eggs preserved in clay and ash for a few months",

"image\_url": "some-valid-url",

"price": "17"

}

}

**Example Response**

{

"data": {

"id": "3c637d011d844ebab1205fef8a7e36ea",

"name": "Century Eggs",

"description": "Whole eggs preserved in clay and ash for a few months",

"image\_url": "some-valid-url",

"price": "17"

}

}

#### GET /orders

This route will respond with a list of all existing order data.

**Example Request**

GET http://localhost:5000/orders

**Example Response**  
Status 200

{

"data": [

{

"id": "5a887d326e83d3c5bdcbee398ea32aff",

"deliverTo": "308 Negra Arroyo Lane, Albuquerque, NM",

"mobileNumber": "(505) 143-3369",

"status": "delivered",

"dishes": [

{

"id": "d351db2b49b69679504652ea1cf38241",

"name": "Dolcelatte and chickpea spaghetti",

"description": "Spaghetti topped with a blend of dolcelatte and fresh chickpeas",

"image\_url": "https://images.pexels.com/photos/1279330/pexels-photo-1279330.jpeg?h=530&w=350",

"price": 19,

"quantity": 2

}

]

}

]

}

#### POST /orders

This route will save the order and respond with the newly created order.

##### Validation:

If any of the following validations fail, respond with a status code of 400 and an error message.

| **Validation** | **Error message** |
| --- | --- |
| deliverTo property is missing | Order must include a deliverTo |
| deliverTo property is empty "" | Order must include a deliverTo |
| mobileNumber property is missing | Order must include a mobileNumber |
| mobileNumber property is empty "" | Order must include a mobileNumber |
| dishes property is missing | Order must include a dish |
| dishes property is not an array | Order must include at least one dish |
| dishes array is empty | Order must include at least one dish |
| a dish quantity property is missing | Dish ${index} must have a quantity that is an integer greater than 0 |
| a dish quantity property is zero or less | Dish ${index} must have a quantity that is an integer greater than 0 |
| a dish quantity property is not an integer | Dish ${index} must have a quantity that is an integer greater than 0 |

**Example Request**

POST http://localhost:5000/orders

Body:

{

"data": {

"deliverTo": "308 Negra Arroyo Lane, Albuquerque, NM",

"mobileNumber": "(505) 143-3369",

"status": "delivered",

"dishes": [

{

"id": "d351db2b49b69679504652ea1cf38241",

"name": "Dolcelatte and chickpea spaghetti",

"description": "Spaghetti topped with a blend of dolcelatte and fresh chickpeas",

"image\_url": "https://images.pexels.com/photos/1279330/pexels-photo-1279330.jpeg?h=530&w=350",

"price": 19,

"quantity": 2

}

]

}

}

**Note**: Each dish in the Order's dishes property is a complete copy of the dish, rather than a reference to the dish by id. This is so the order does not change retroactively if the dish data is updated some time after the order is created.

**Example Response**

Status 201

{

"data": {

"id": "5a887d326e83d3c5bdcbee398ea32aff",

"deliverTo": "308 Negra Arroyo Lane, Albuquerque, NM",

"mobileNumber": "(505) 143-3369",

"status": "delivered",

"dishes": [

{

"id": "d351db2b49b69679504652ea1cf38241",

"name": "Dolcelatte and chickpea spaghetti",

"description": "Spaghetti topped with a blend of dolcelatte and fresh chickpeas",

"image\_url": "https://images.pexels.com/photos/1279330/pexels-photo-1279330.jpeg?h=530&w=350",

"price": 19,

"quantity": 2

}

]

}

}

#### GET /orders/:orderId

This route will respond with the order where id === :orderId, or return 404 if no matching order is found.

**Example Request**

GET http://localhost:5000/orders/f6069a542257054114138301947672ba

**Example Response**

Status 200

{

"data": {

"id": "f6069a542257054114138301947672ba",

"deliverTo": "1600 Pennsylvania Avenue NW, Washington, DC 20500",

"mobileNumber": "(202) 456-1111",

"status": "out-for-delivery",

"dishes": [

{

"id": "90c3d873684bf381dfab29034b5bba73",

"name": "Falafel and tahini bagel",

"description": "A warm bagel filled with falafel and tahini",

"image\_url": "https://images.pexels.com/photos/4560606/pexels-photo-4560606.jpeg?h=530&w=350",

"price": 6,

"quantity": 1

}

]

}

}

#### PUT /orders/:orderId

This route will update the order where id === :orderId, or return 404 if no matching order is found.

##### Validation:

The update validation must include all of the same validation as the POST /orders route, plus the following:

| **Validation** | **Error message** |
| --- | --- |
| id of body does not match :orderId from the route | Order id does not match route id. Order: ${id}, Route: ${orderId}. |
| status property is missing | Order must have a status of pending, preparing, out-for-delivery, delivered |
| status property is empty | Order must have a status of pending, preparing, out-for-delivery, delivered |
| status property of the existing order === "delivered" | A delivered order cannot be changed |

Note: The id property is not required in the body of the request, but if it is present it must match :orderId from the route.

**Example Request**

PUT http://localhost:5000/orders/3c637d011d844ebab1205fef8a7e36ea

Body:

{

"data": {

"deliverTo": "Rick Sanchez (C-132)",

"mobileNumber": "(202) 456-1111",

"status": "delivered",

"dishes": [

{

"id": "90c3d873684bf381dfab29034b5bba73",

"name": "Falafel and tahini bagel",

"description": "A warm bagel filled with falafel and tahini",

"image\_url": "https://images.pexels.com/photos/4560606/pexels-photo-4560606.jpeg?h=530&w=350",

"price": 6,

"quantity": 1

}

]

}

}

**Example Response**

{

"data": {

"id": "f6069a542257054114138301947672ba",

"deliverTo": "Rick Sanchez (C-132)",

"mobileNumber": "(202) 456-1111",

"status": "delivered",

"dishes": [

{

"id": "90c3d873684bf381dfab29034b5bba73",

"name": "Falafel and tahini bagel",

"description": "A warm bagel filled with falafel and tahini",

"image\_url": "https://images.pexels.com/photos/4560606/pexels-photo-4560606.jpeg?h=530&w=350",

"price": 6,

"quantity": 1

}

]

}

}

#### DELETE /orders/:orderId

This route will delete the order where id === :orderId, or return 404 if no matching order is found.

##### Validation:

The delete method must include the following validation:

| **Validation** | **Error message** |
| --- | --- |
| status property of the order !== "pending" | An order cannot be deleted unless it is pending |

**Example Request**

DELETE http://localhost:5000/dishes/3c637d011d844ebab1205fef8a7e36ea

**Example Response**  
Status 204 and no response body.

**Note:** In addition to needing to pass the tests and requirements in the instructions here, please review the Rubric Requirements for the human-graded part of this project in your Thinkful curriculum page.