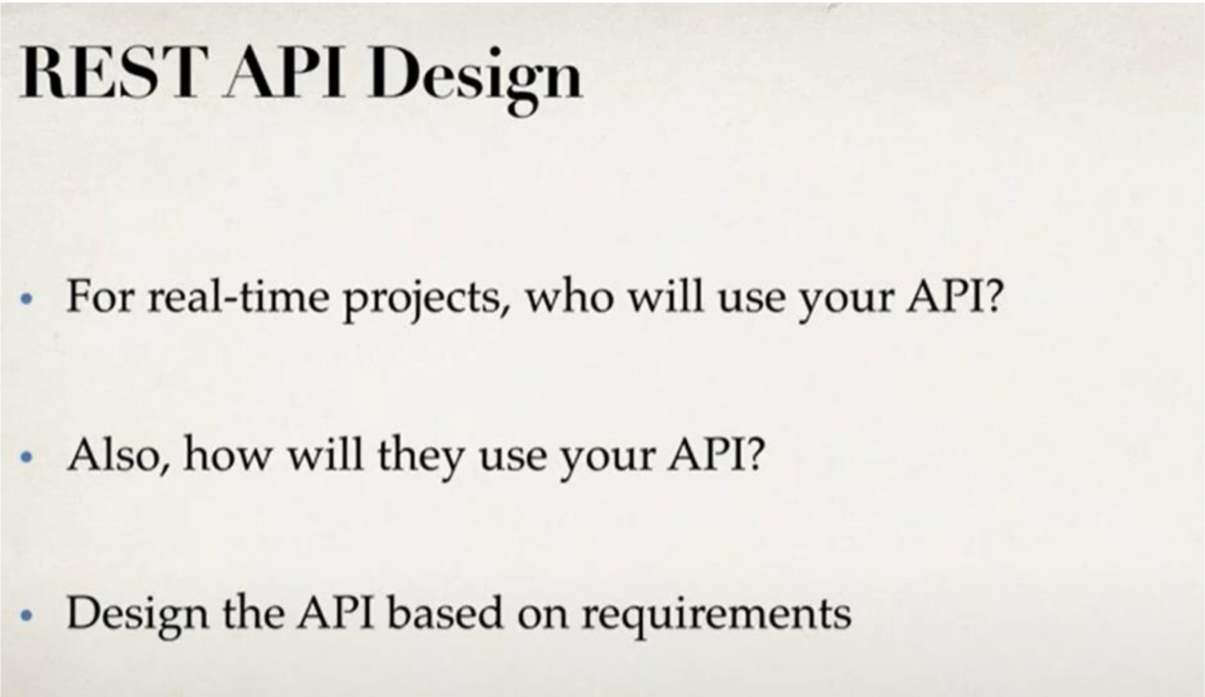
Spring Rest API Design:

Spring REST API Design



REST API Design

- For real-time projects, who will use your API?
- Also, how will they use your API?
- Design the API based on requirements

API Design Process

Step-By-Step

1. Review API requirements

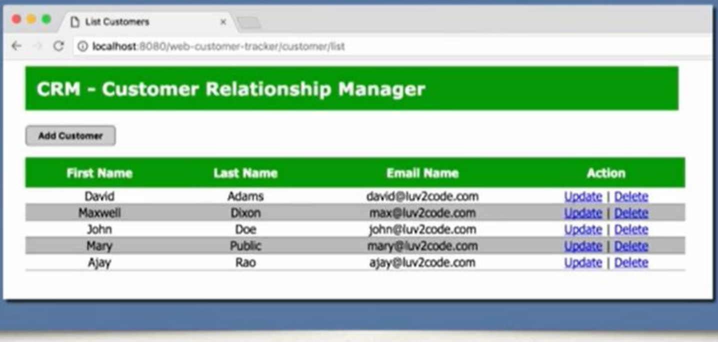
2. Identify main resource / entity

3. Use HTTP methods to assign action on resource

Step 1: Review API Requirements

From the Boss

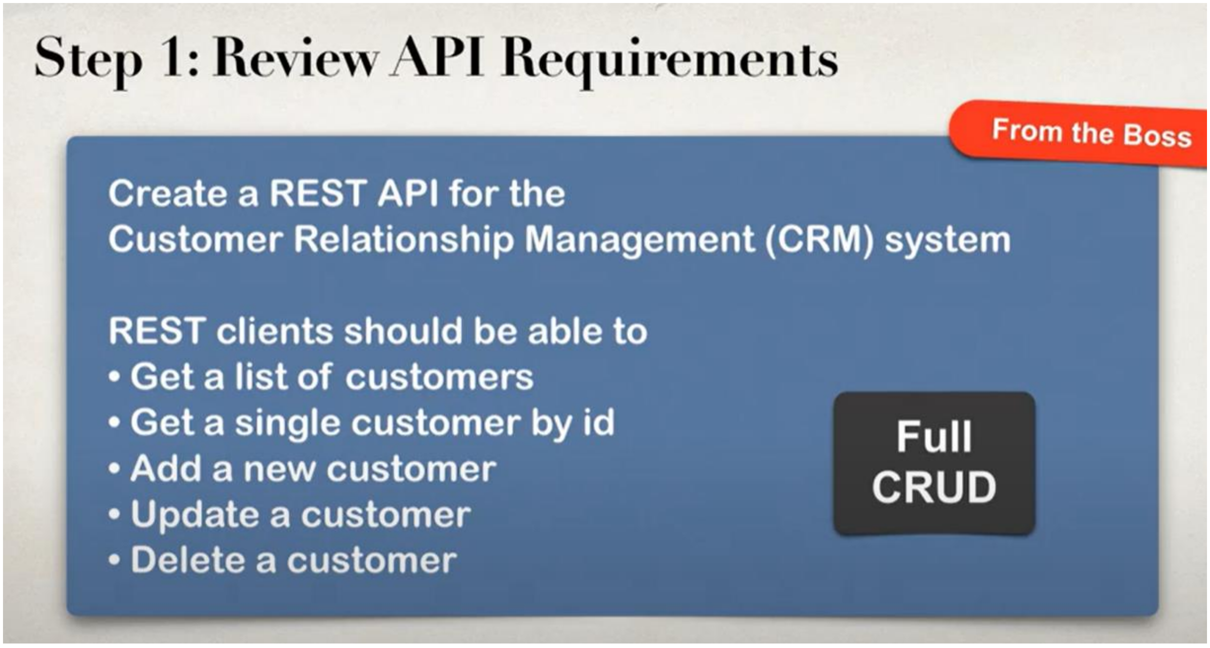
Create a REST API for the
Customer Relationship Management (CRM) system



CRM - Customer Relationship Manager

Add Customer

First Name	Last Name	Email Name	Action
David	Adams	david@luv2code.com	Update | Delete
Maxwell	Dixon	max@luv2code.com	Update | Delete
John	Doe	john@luv2code.com	Update | Delete
Mary	Public	mary@luv2code.com	Update | Delete
Ajay	Rao	ajay@luv2code.com	Update | Delete



Step 1: Review API Requirements

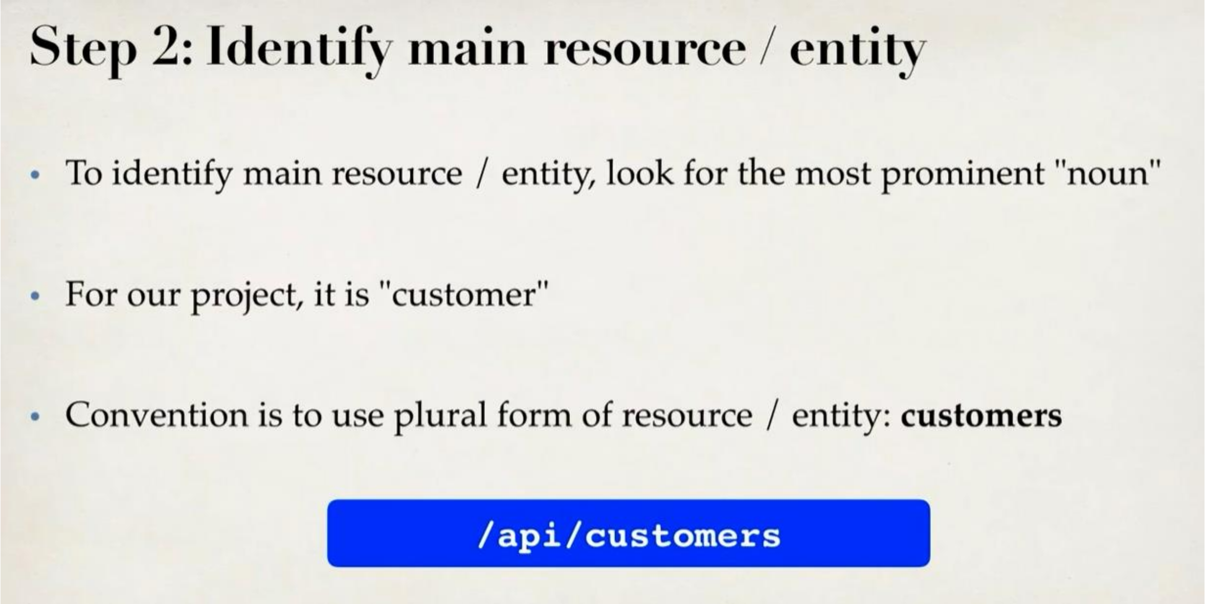
From the Boss

Create a REST API for the
Customer Relationship Management (CRM) system

REST clients should be able to

- Get a list of customers
- Get a single customer by id
- Add a new customer
- Update a customer
- Delete a customer

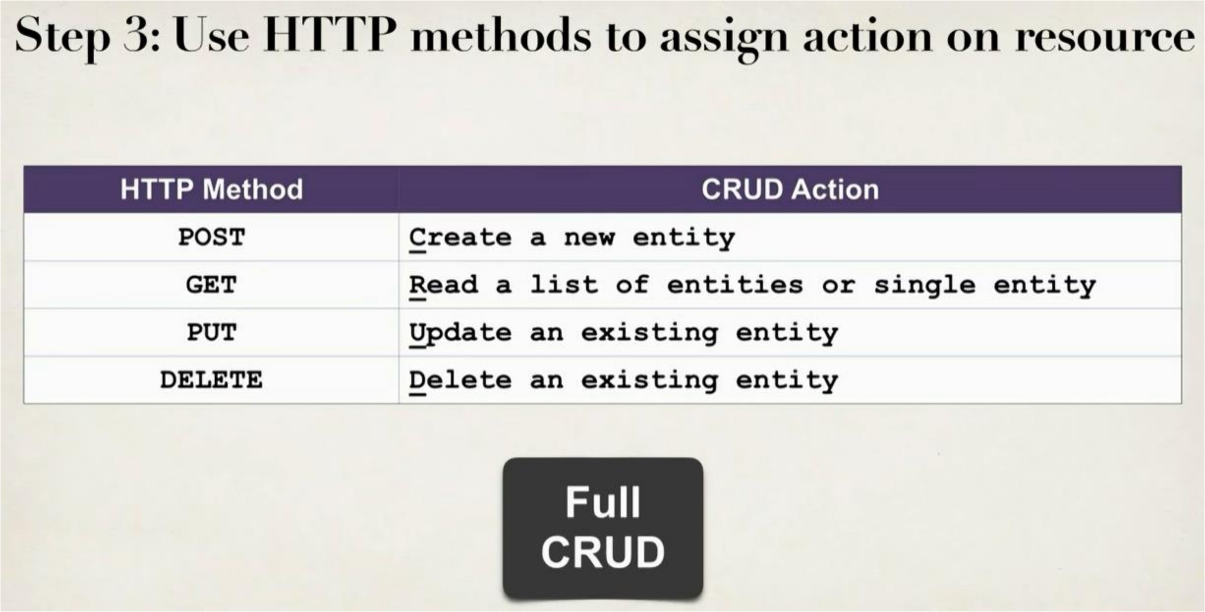
Full CRUD



Step 2: Identify main resource / entity

- To identify main resource / entity, look for the most prominent "noun"
- For our project, it is "customer"
- Convention is to use plural form of resource / entity: **customers**

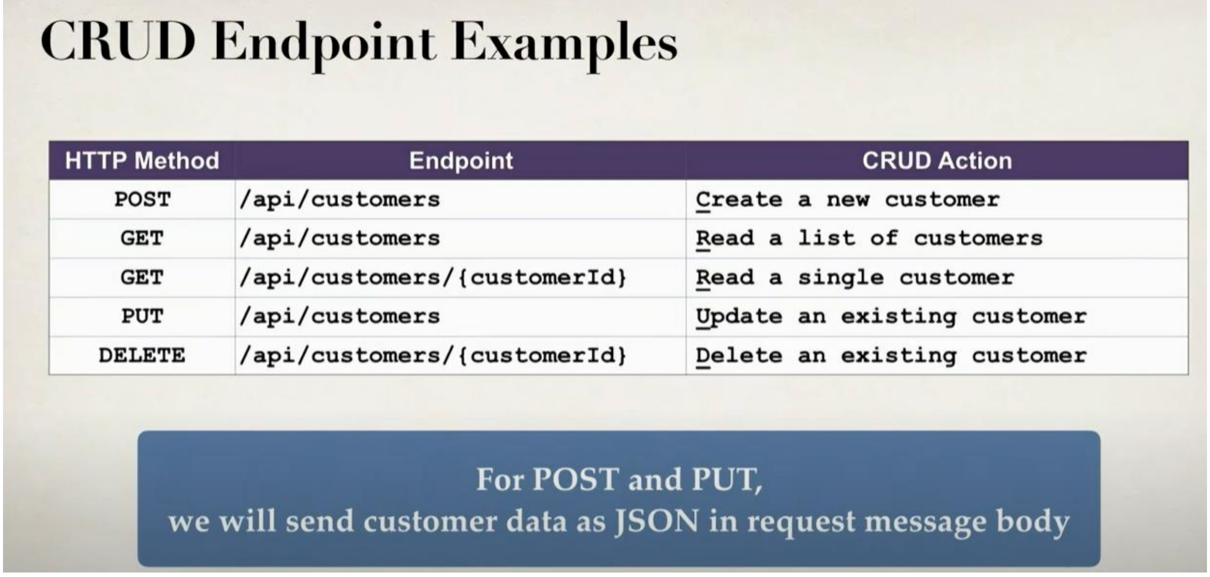
```
/api/customers
```



Step 3: Use HTTP methods to assign action on resource

HTTP Method	CRUD Action
POST	Create a new entity
GET	Read a list of entities or single entity
PUT	Update an existing entity
DELETE	Delete an existing entity

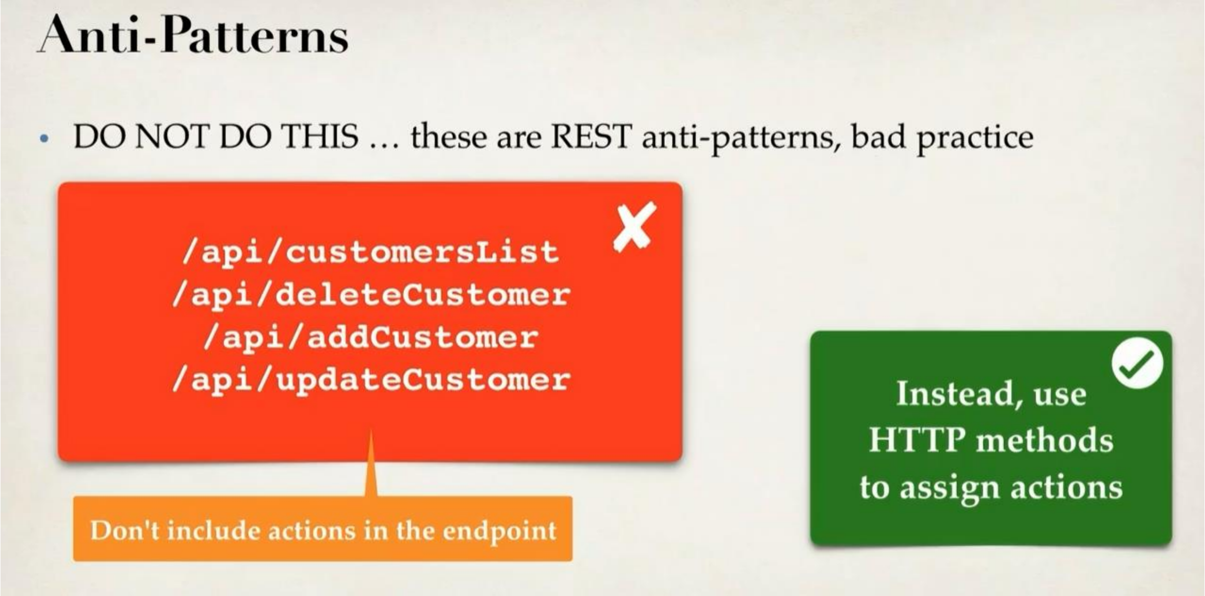
Full CRUD



CRUD Endpoint Examples

HTTP Method	Endpoint	CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

For POST and PUT,
we will send customer data as JSON in request message body



Anti-Patterns

- DO NOT DO THIS … these are REST anti-patterns, bad practice

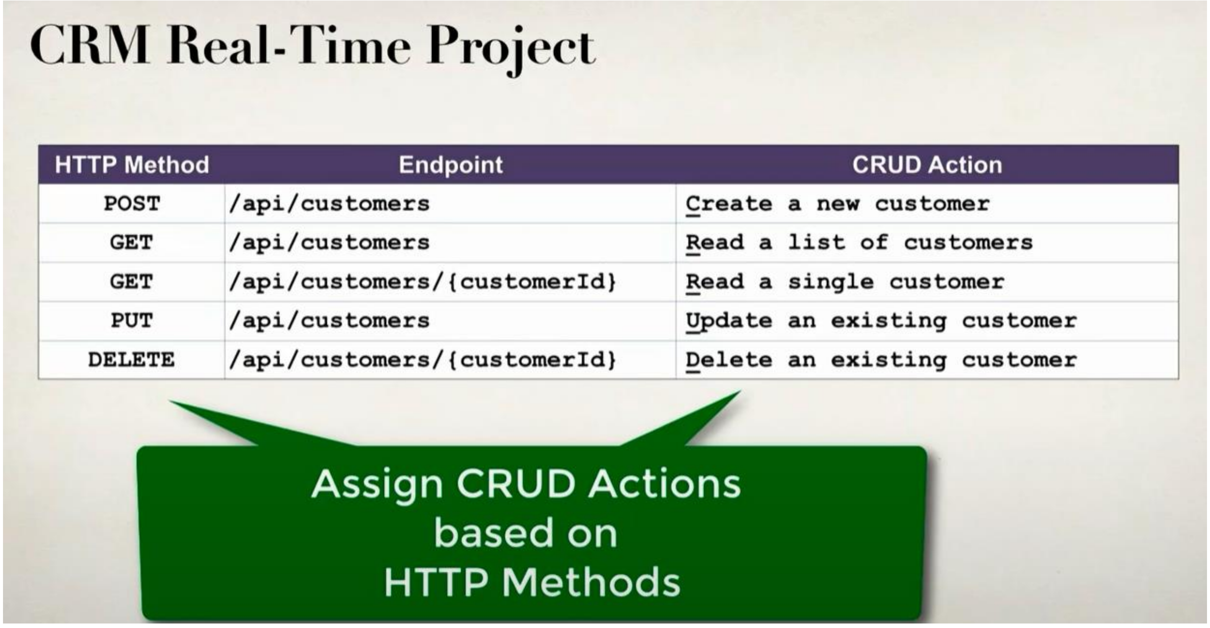
```
/api/customersList
/api/deleteCustomer
/api/addCustomer
/api/updateCustomer
```

✗

Don't include actions in the endpoint

Instead, use
HTTP methods
to assign actions

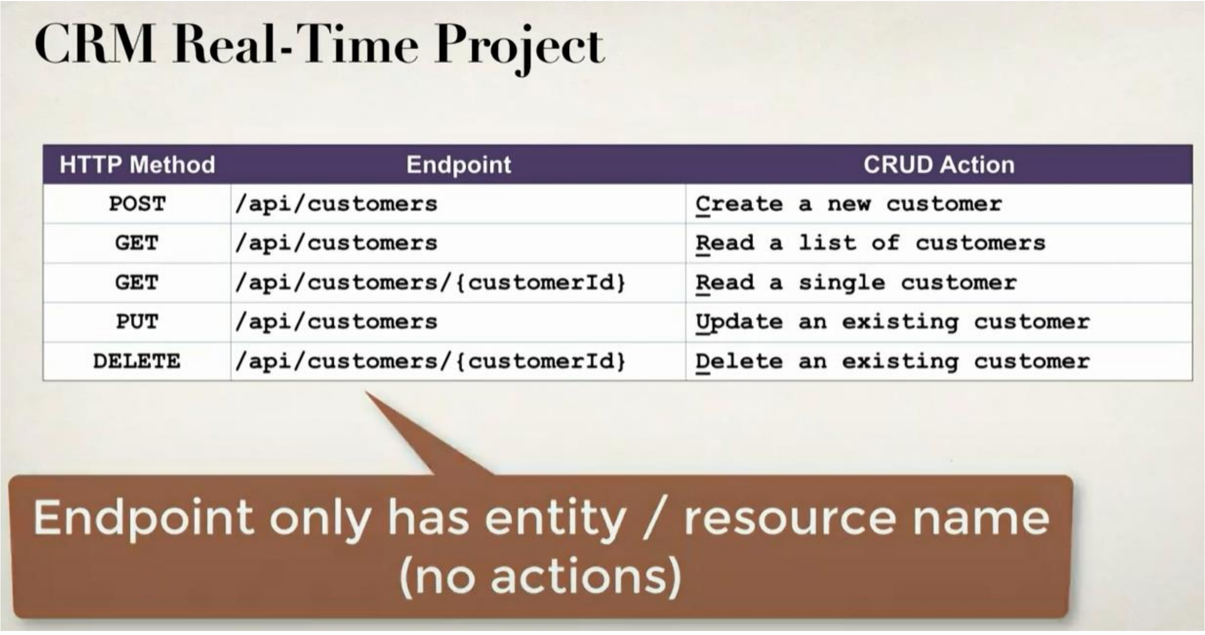
✓



CRM Real-Time Project

HTTP Method	Endpoint	CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

Assign CRUD Actions
based on
HTTP Methods



CRM Real-Time Project

HTTP Method	Endpoint	CRUD Action
POST	/api/customers	Create a new customer
GET	/api/customers	Read a list of customers
GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer

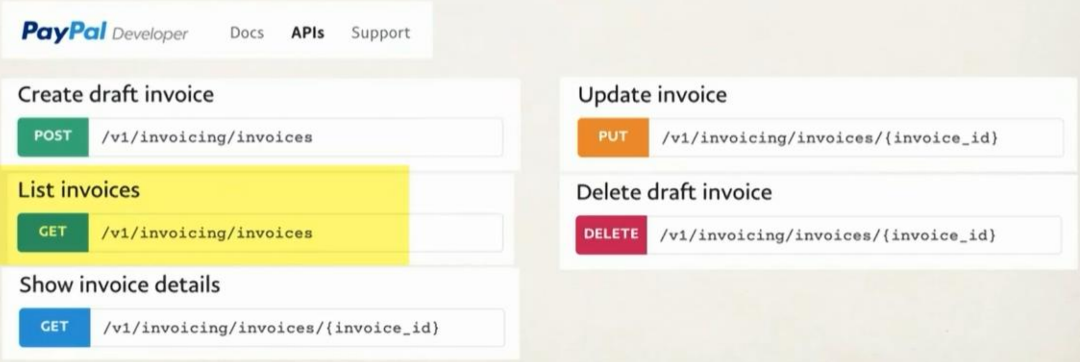
Endpoint only has entity / resource name
(no actions)

More API Examples

- On the following slides, we'll look at APIs from other real-time projects
- PayPal
- GitHub
- SalesForce

PayPal

- PayPal Invoicing API
  - `https://developer.paypal.com/docs/api/invoicing/`



Create draft invoice
POST `/v1/invoicing/invoices`

List invoices
GET `/v1/invoicing/invoices`

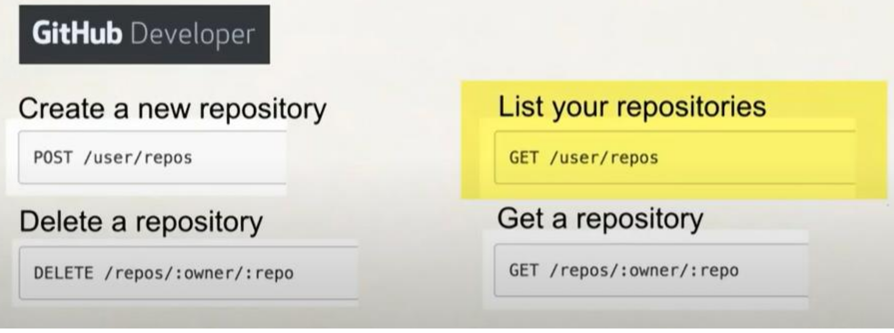
Show invoice details
GET `/v1/invoicing/invoices/{invoice_id}`

Update invoice
PUT `/v1/invoicing/invoices/{invoice_id}`

Delete draft invoice
DELETE `/v1/invoicing/invoices/{invoice_id}`

GitHub

- GitHub Repositories API
  - `https://developer.github.com/v3/repos/#repositories`

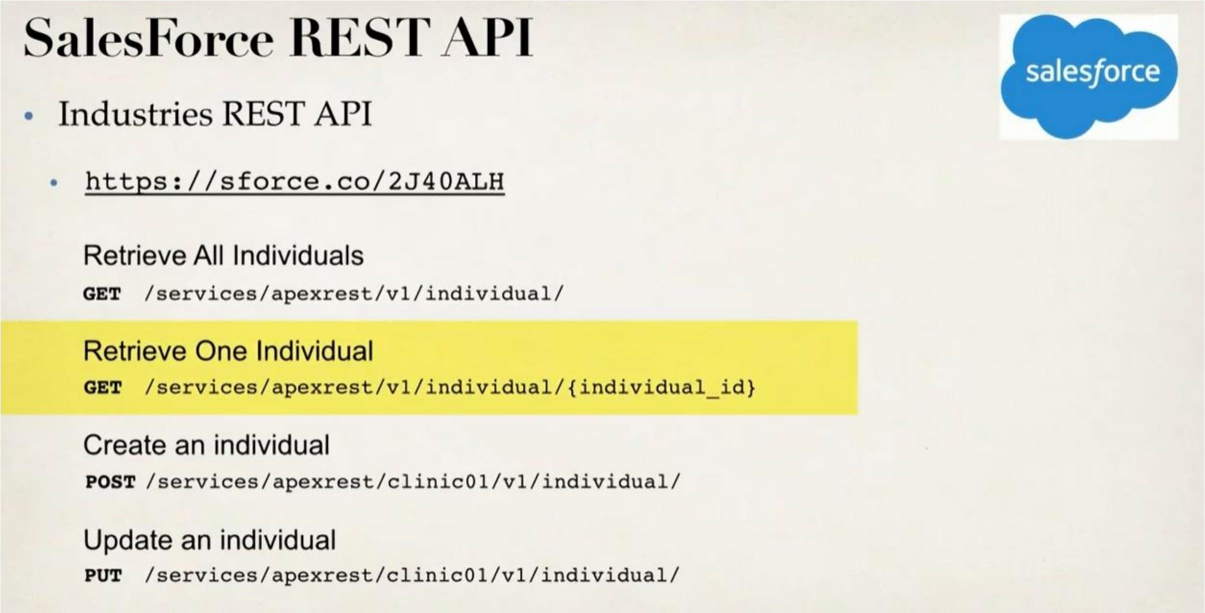


Create a new repository
`POST /user/repos`

Delete a repository
`DELETE /repos/:owner/:repo`

List your repositories
`GET /user/repos`

Get a repository
`GET /repos/:owner/:repo`



SalesForce REST API

- Industries REST API
  - https://sforce.co/2J40ALH

Retrieve All Individuals

```
GET  /services/apexrest/v1/individual/
```

Retrieve One Individual

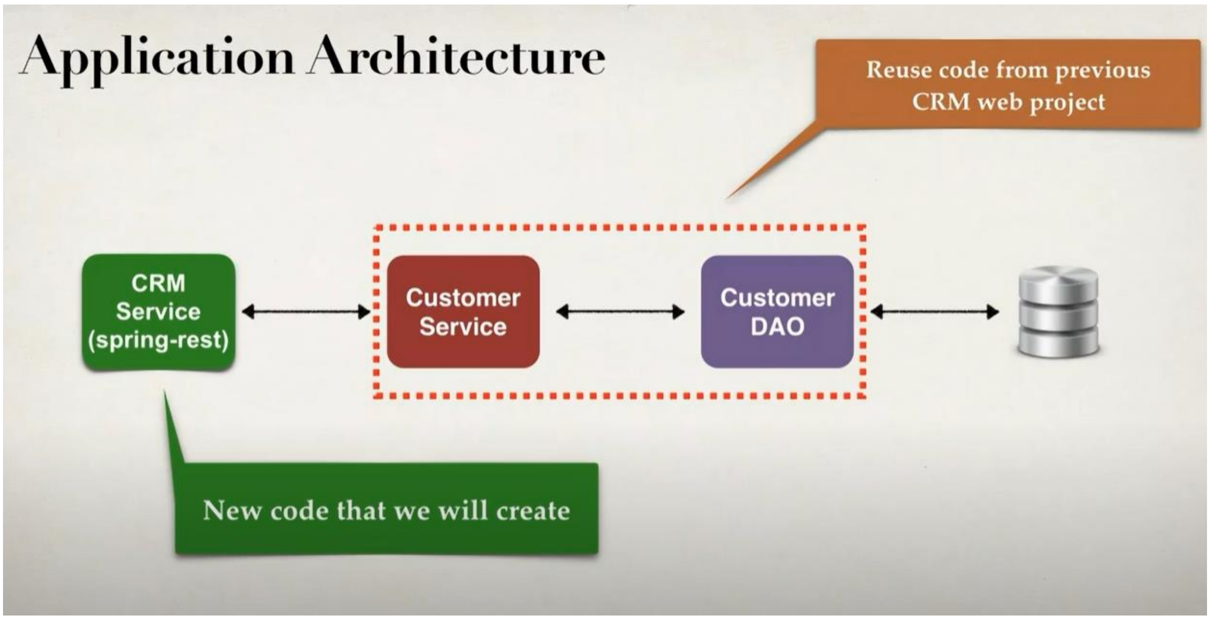
```
GET  /services/apexrest/v1/individual/{individual_id}
```

Create an individual

```
POST /services/apexrest/clinic01/v1/individual/
```

Update an individual

```
PUT  /services/apexrest/clinic01/v1/individual/
```



Application Architecture

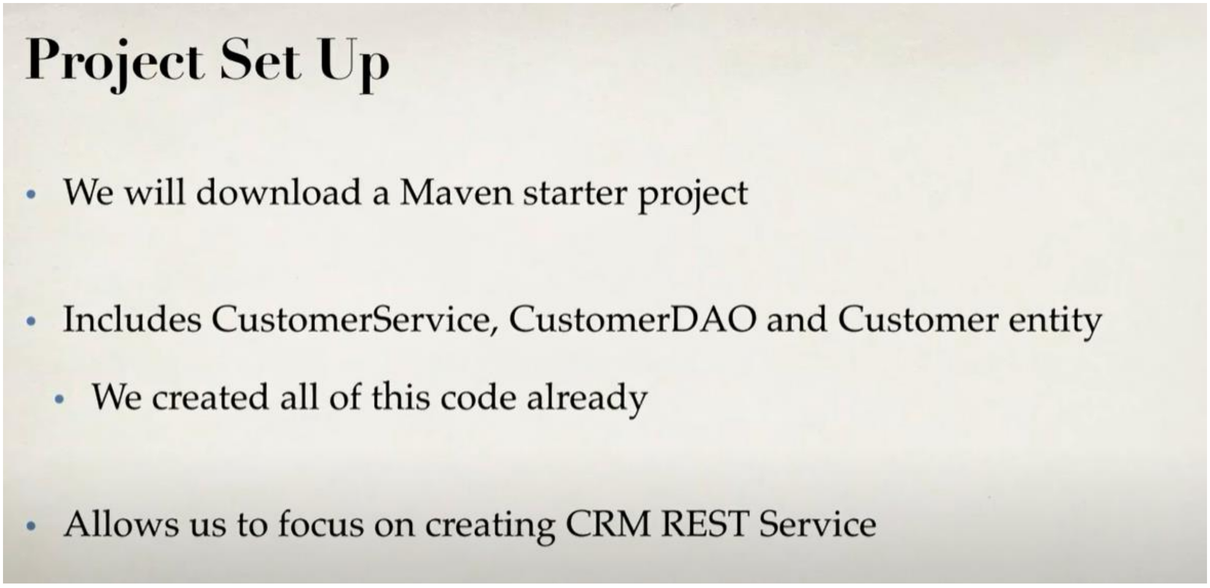
Reuse code from previous CRM web project

CRM Service (spring-rest)

Customer Service

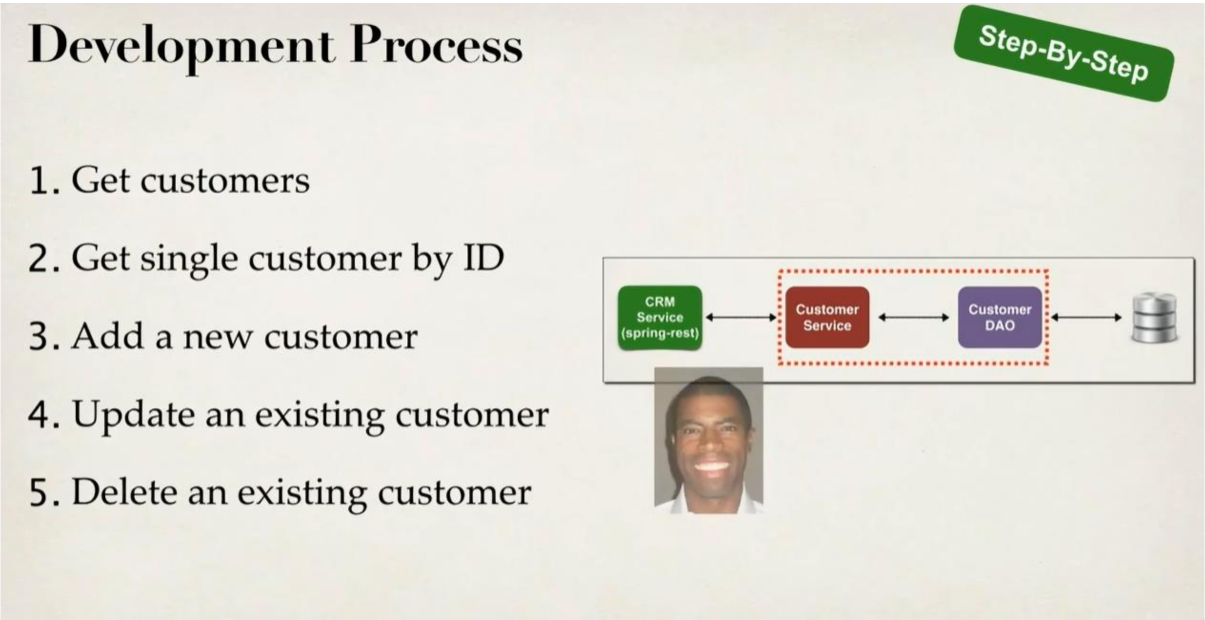
Customer DAO

New code that we will create



Project Set Up

- We will download a Maven starter project
- Includes CustomerService, CustomerDAO and Customer entity
  - We created all of this code already
- Allows us to focus on creating CRM REST Service

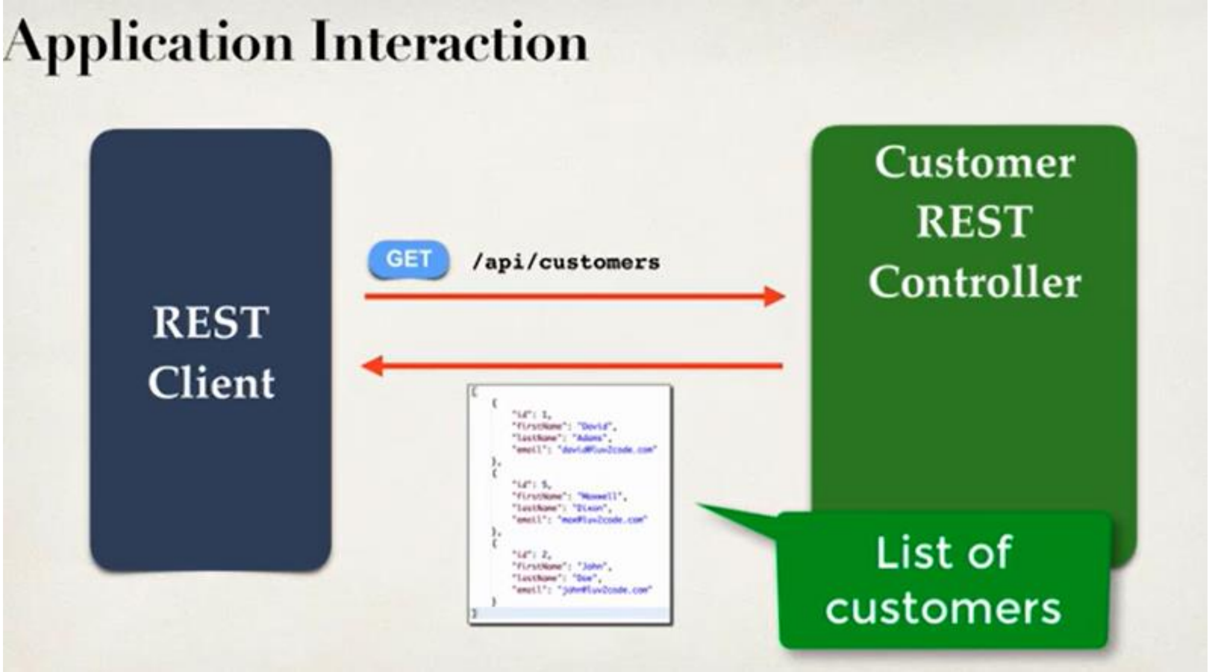


Development Process

Step-By-Step

1. Get customers
2. Get single customer by ID
3. Add a new customer
4. Update an existing customer
5. Delete an existing customer

Spring CRM REST - Get Customers



Application Interaction

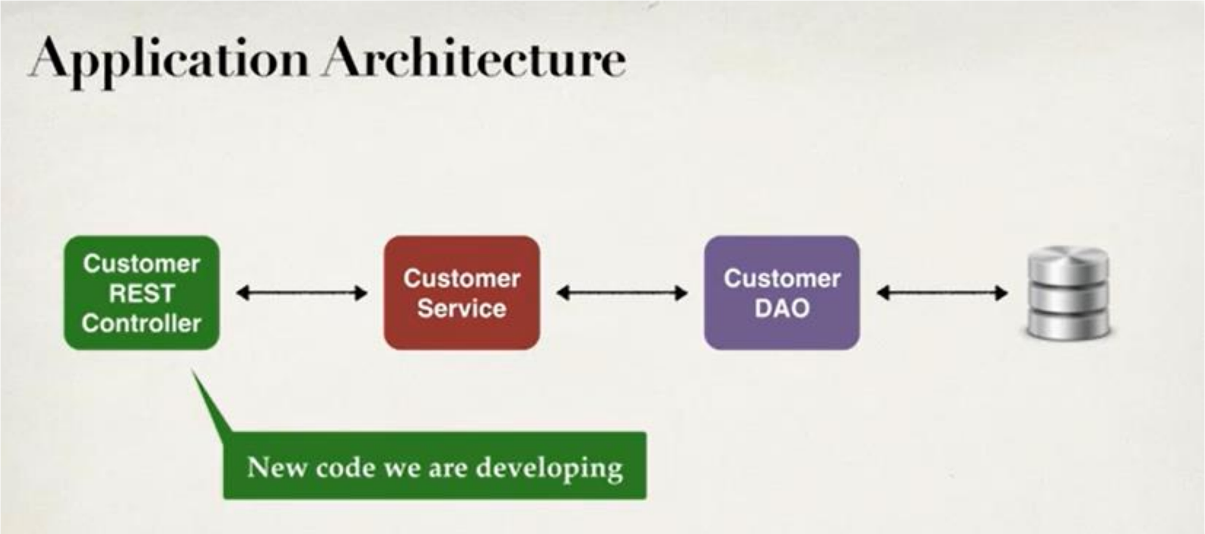
REST Client

Customer REST Controller

GET /api/customers

```
[
    {
        "id": 1,
        "firstName": "David",
        "lastName": "Adams",
        "email": "david@luv2code.com"
    },
    {
        "id": 5,
        "firstName": "Maxwell",
        "lastName": "Dixon",
        "email": "max@luv2code.com"
    },
    {
        "id": 2,
        "firstName": "John",
        "lastName": "Doe",
        "email": "john@luv2code.com"
    }
]
```

List of customers



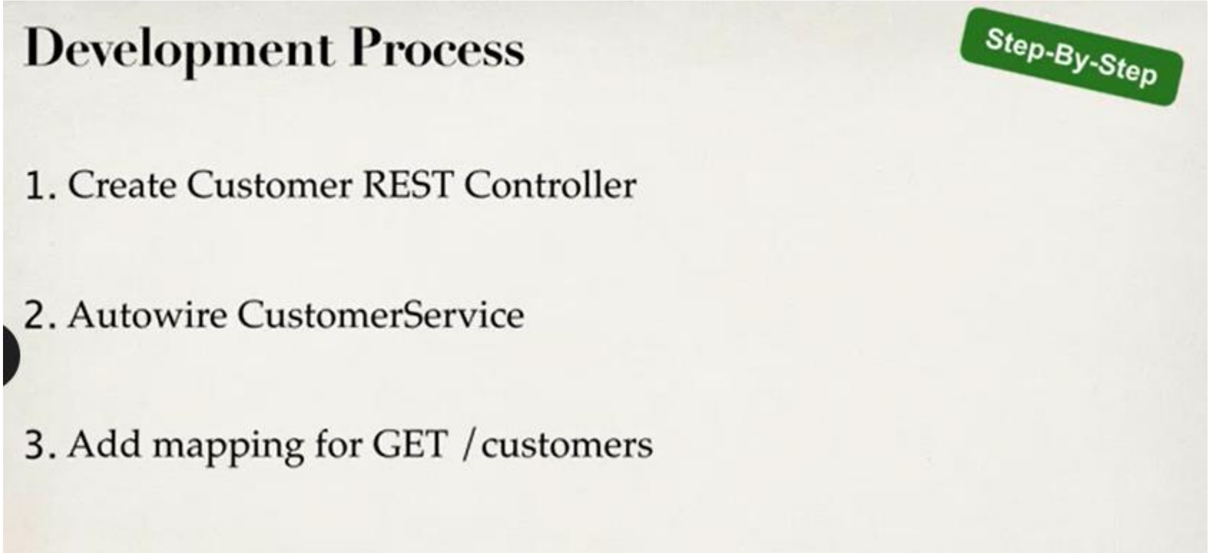
Application Architecture

Customer REST Controller

Customer Service

Customer DAO

New code we are developing



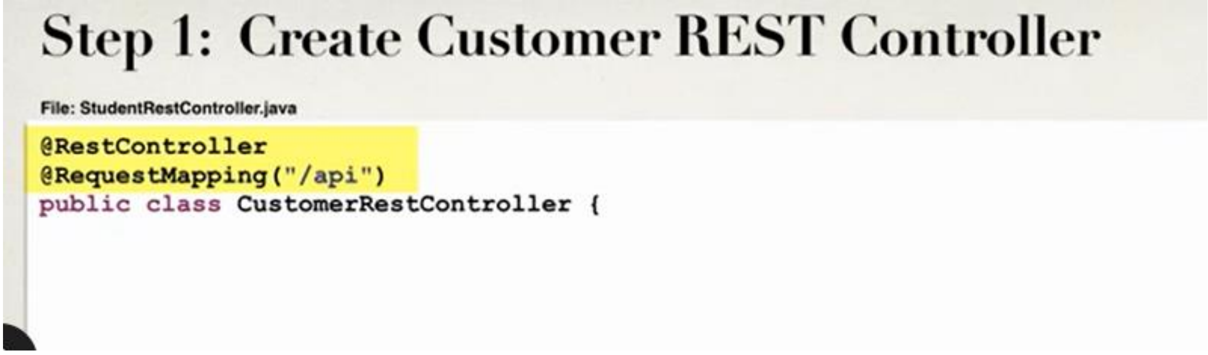
Development Process

Step-By-Step

1. Create Customer REST Controller

2. Autowire CustomerService

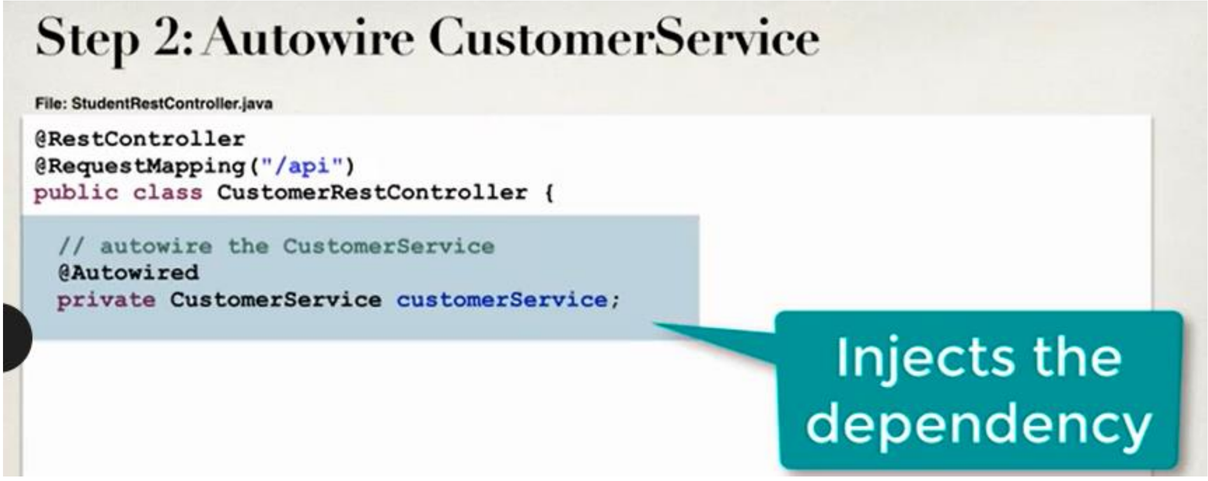
3. Add mapping for GET /customers



Step 1: Create Customer REST Controller

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
```

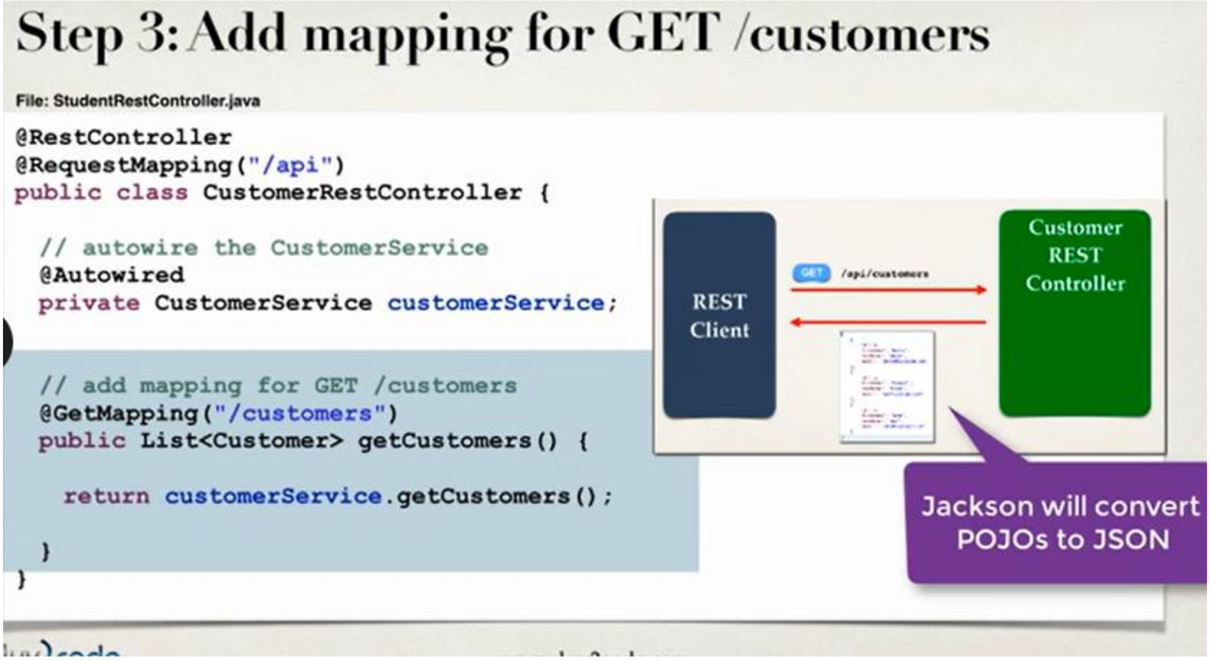


Step 2: Autowire CustomerService

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {

  // autowire the CustomerService
  @Autowired
  private CustomerService customerService;
```

Injects the dependency

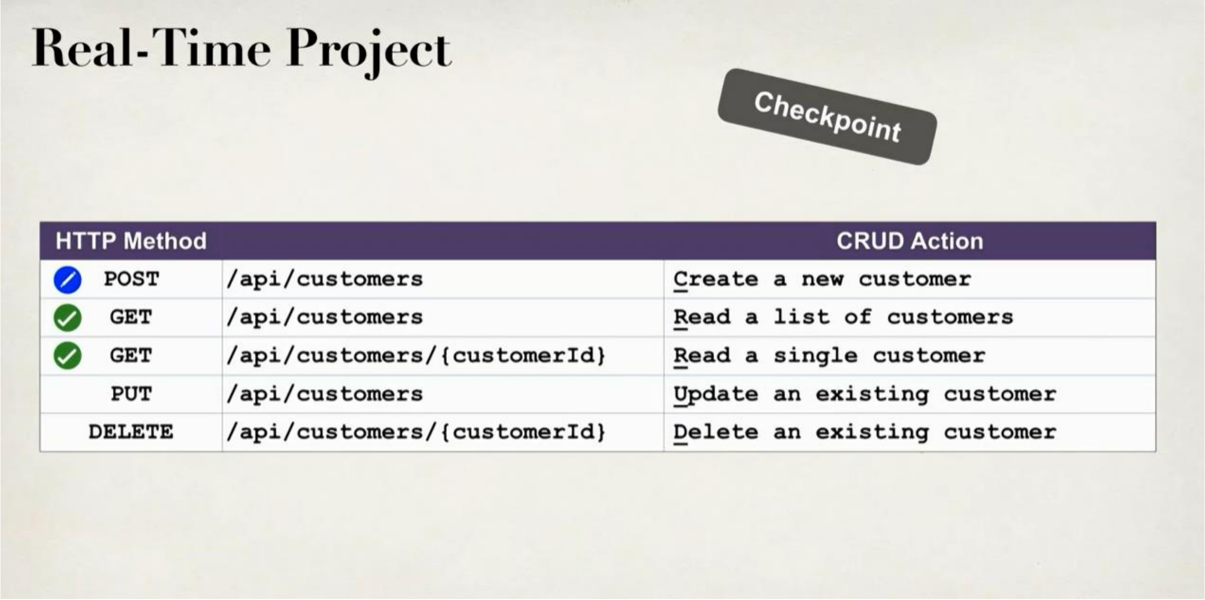


Step 3: Add mapping for GET /customers

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {

  // autowire the CustomerService
  @Autowired
  private CustomerService customerService;

  // add mapping for GET /customers
  @GetMapping("/customers")
  public List<Customer> getCustomers() {

    return customerService.getCustomers();

  }
}
```

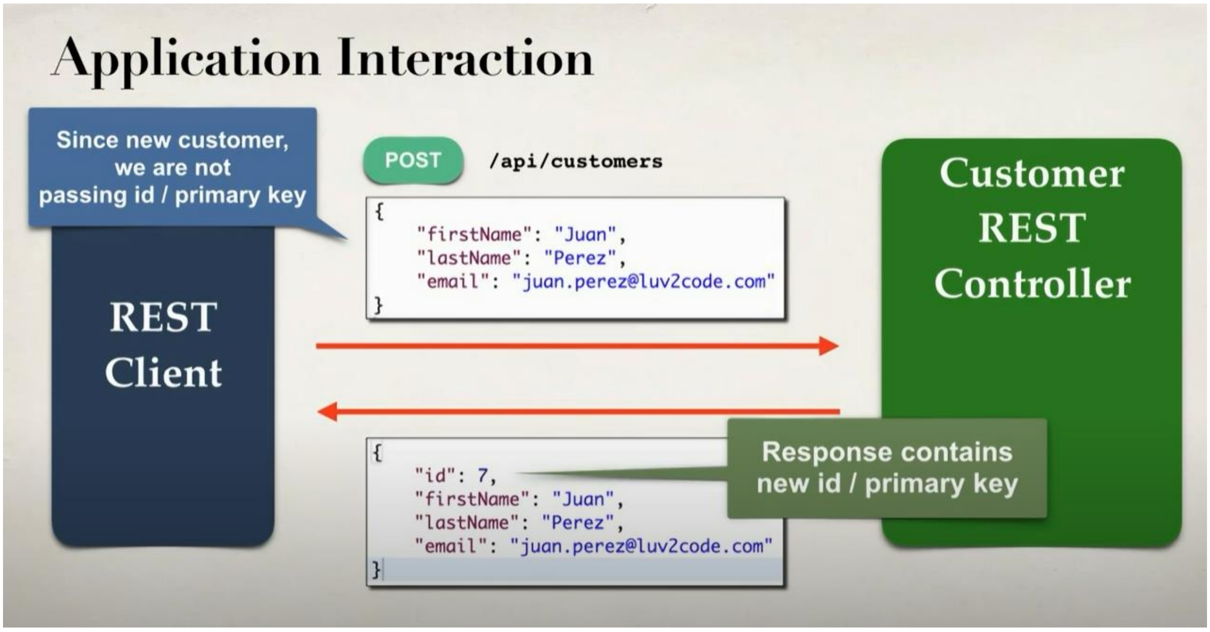
Jackson will convert POJOs to JSON



Real-Time Project

Checkpoint

HTTP Method		CRUD Action
✎ POST	/api/customers	Create a new customer
✓ GET	/api/customers	Read a list of customers
✓ GET	/api/customers/{customerId}	Read a single customer
PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer



Application Interaction

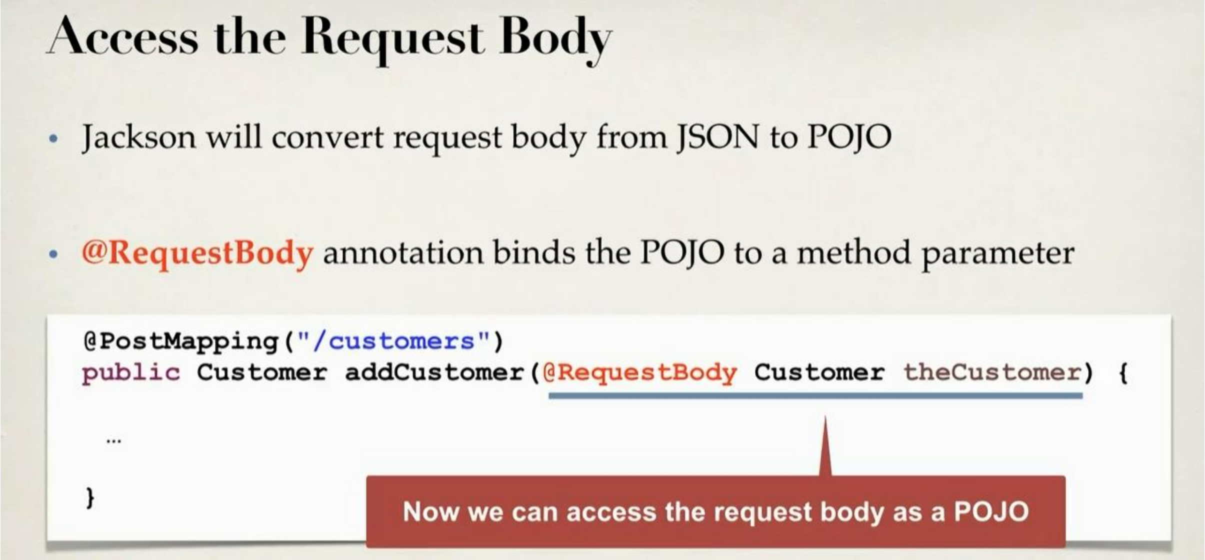
Since new customer, we are not passing id / primary key

POST /api/customers

```
{
    "firstName": "Juan",
    "lastName": "Perez",
    "email": "juan.perez@luv2code.com"
}
```

```
{
    "id": 7,
    "firstName": "Juan",
    "lastName": "Perez",
    "email": "juan.perez@luv2code.com"
}
```

Response contains new id / primary key

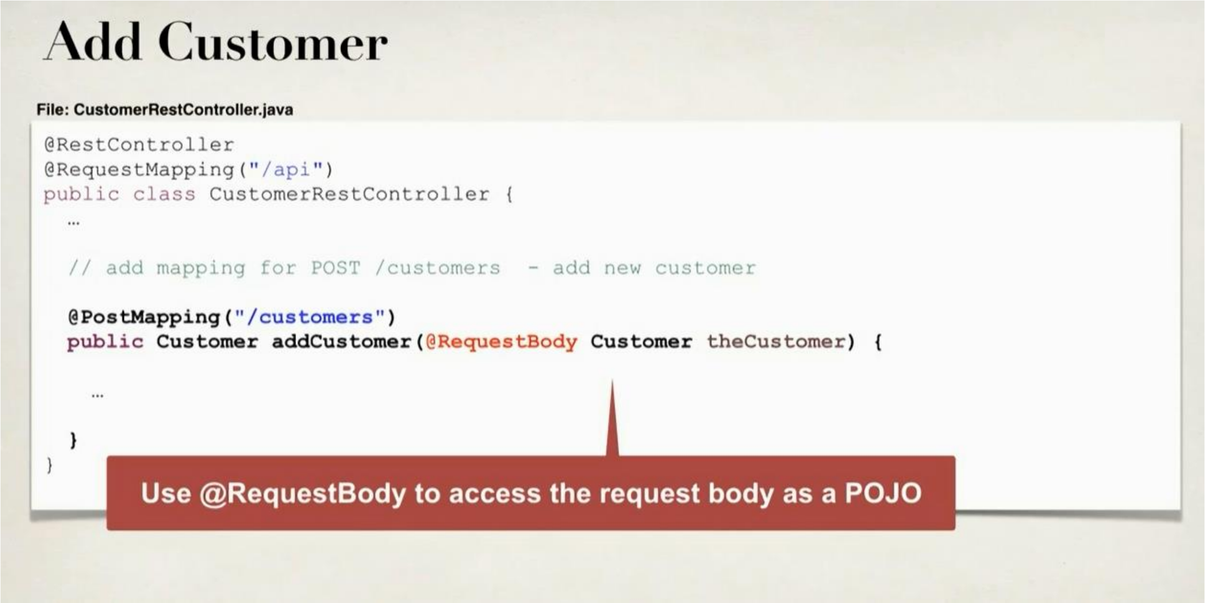


Access the Request Body

- Jackson will convert request body from JSON to POJO
- **@RequestBody** annotation binds the POJO to a method parameter

```
@PostMapping("/customers")
public Customer addCustomer(@RequestBody Customer theCustomer) {

  ...

}
```

Now we can access the request body as a POJO

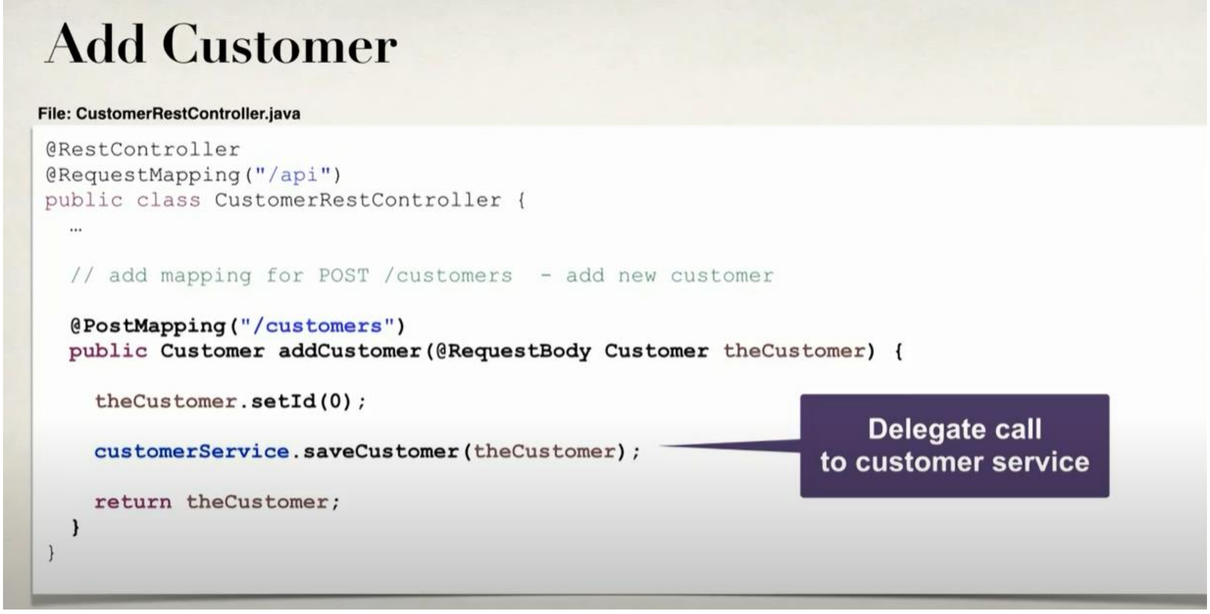


Add Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for POST /customers  - add new customer

  @PostMapping("/customers")
  public Customer addCustomer(@RequestBody Customer theCustomer) {

    …

  }
}
```

Use @RequestBody to access the request body as a POJO



Add Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  …

  // add mapping for POST /customers  - add new customer

  @PostMapping("/customers")
  public Customer addCustomer(@RequestBody Customer theCustomer) {

    theCustomer.setId(0);

    customerService.saveCustomer(theCustomer);

    return theCustomer;
  }
}
```

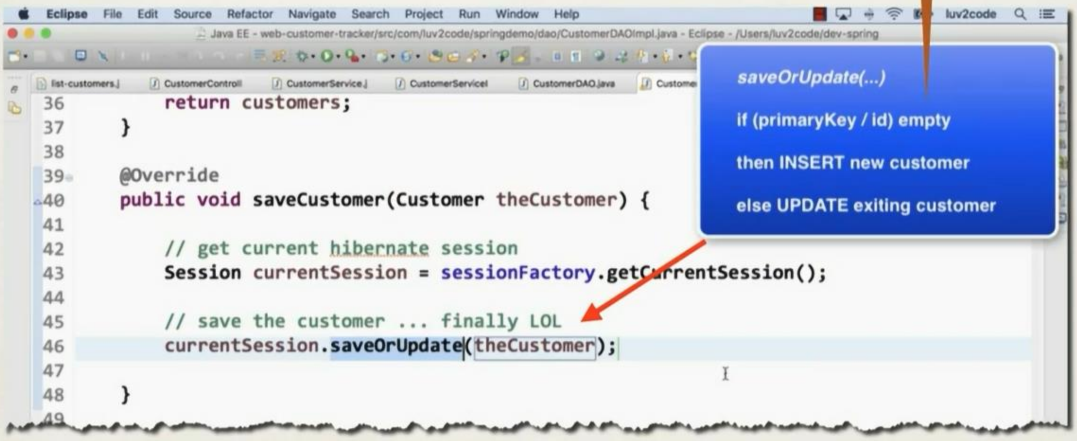
Delegate call
to customer service

What's up with customer id?

- In the REST controller, we explicitly set the customer id to 0
- Because our backend DAO code uses Hibernate method
  - `session.saveOrUpdate(….)`

Recall: CustomerDAOImpl

Here: "empty" means null or 0



Eclipse File Edit Source Refactor Navigate Search Project Run Window Help
luv2code
Java EE - web-customer-tracker/src/com/luv2code/springdemo/dao/CustomerDAOImpl.java - Eclipse - /Users/luv2code/dev-spring
list-customers.j
CustomerControll
CustomerService.j
CustomerServiceI
CustomerDAO.java
Custome

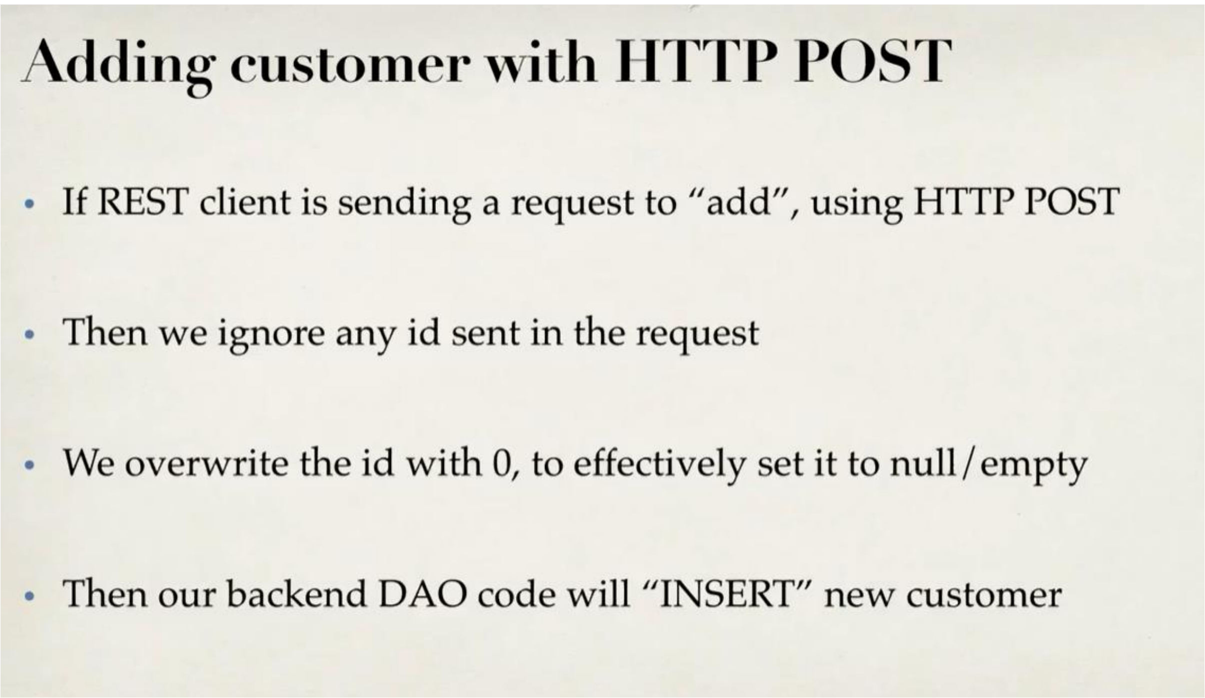
```
        return customers;
    }

    @Override
    public void saveCustomer(Customer theCustomer) {

        // get current hibernate session
        Session currentSession = sessionFactory.getCurrentSession();

        // save the customer ... finally LOL
        currentSession.saveOrUpdate(theCustomer);

    }
```

saveOrUpdate(…)

if (primaryKey / id) empty

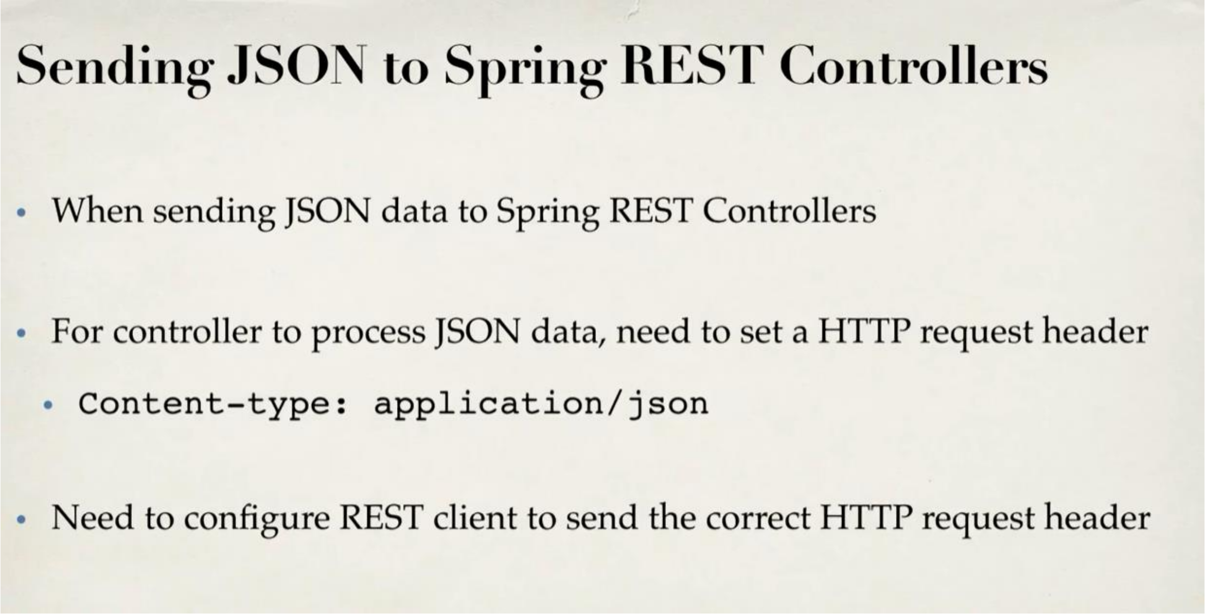
then INSERT new customer

else UPDATE exiting customer



Adding customer with HTTP POST

- If REST client is sending a request to "add", using HTTP POST
- Then we ignore any id sent in the request
- We overwrite the id with 0, to effectively set it to null/empty
- Then our backend DAO code will "INSERT" new customer

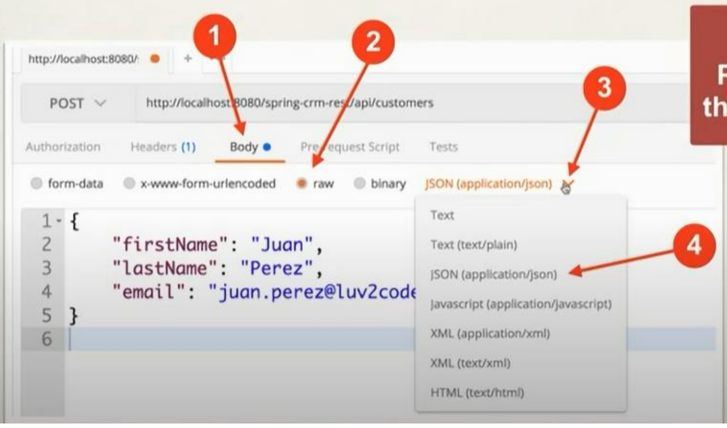


Sending JSON to Spring REST Controllers

- When sending JSON data to Spring REST Controllers
- For controller to process JSON data, need to set a HTTP request header
  - `Content-type: application/json`
- Need to configure REST client to send the correct HTTP request header

Postman - Sending JSON in Request Body

- Must set HTTP request header in Postman

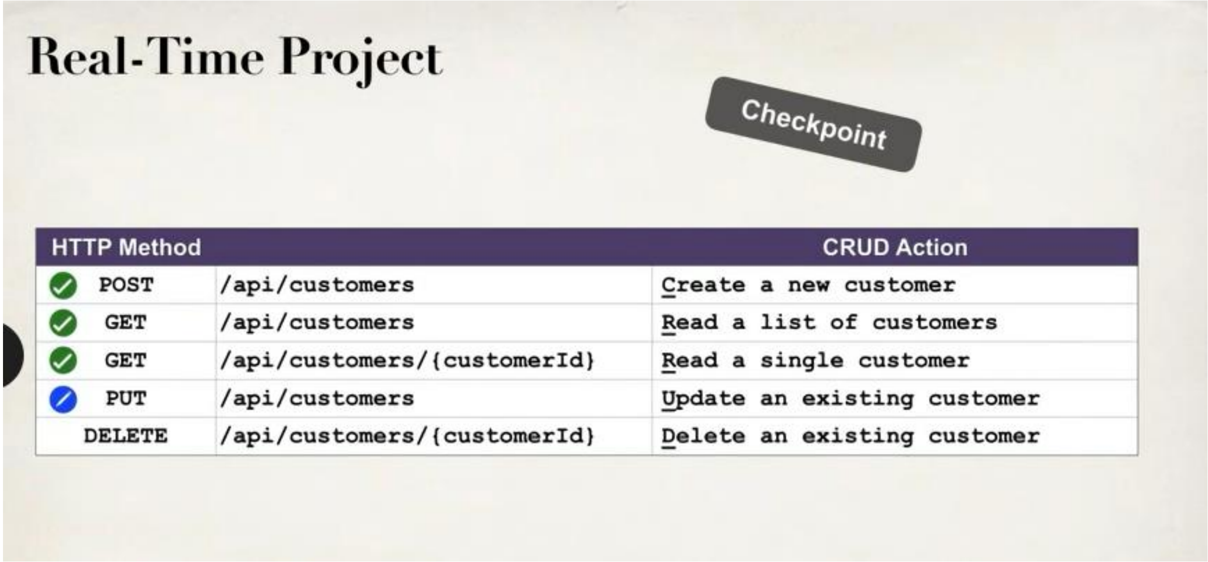


http://localhost:8080/spring-crm-rest/api/customers

```
{
    "firstName": "Juan",
    "lastName": "Perez",
    "email": "juan.perez@luv2code
}
```

Based on these configs, Postman will automatically set the correct HTTP request header

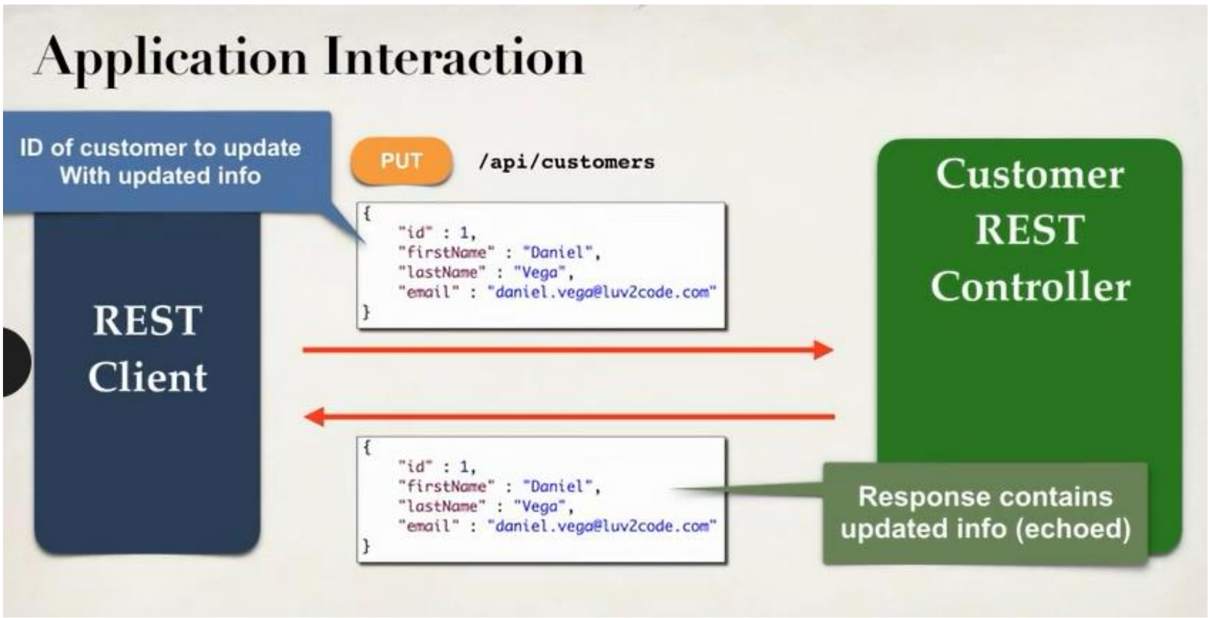
Spring CRM REST - Update Customer



Real-Time Project

Checkpoint

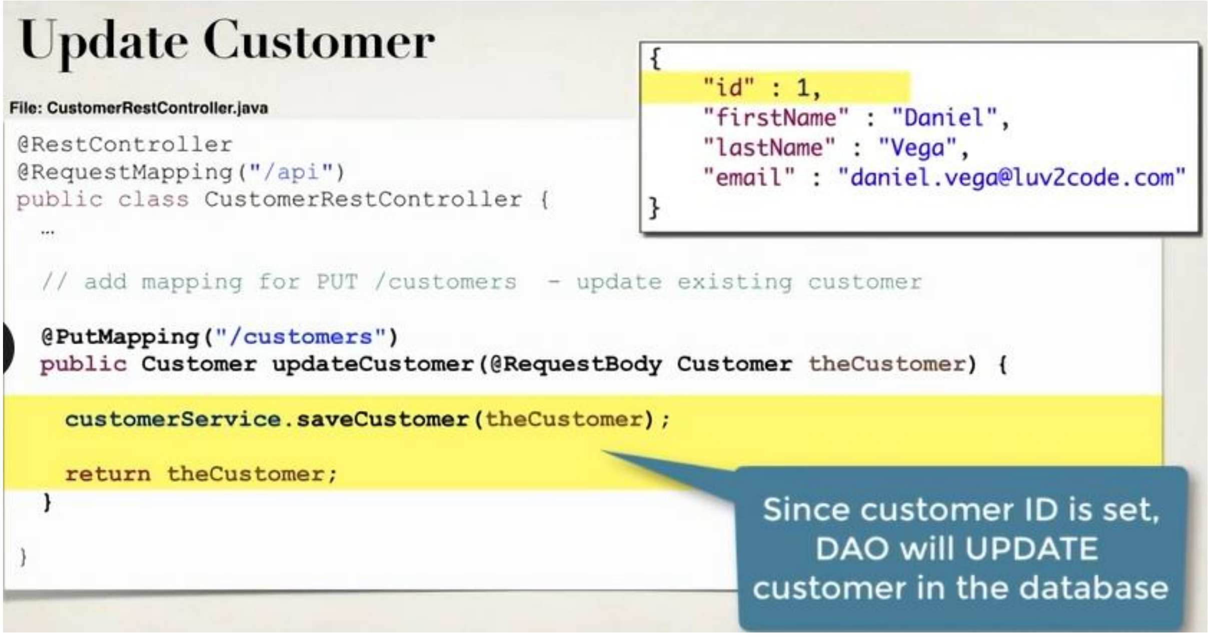
HTTP Method		CRUD Action
✓ POST	/api/customers	Create a new customer
✓ GET	/api/customers	Read a list of customers
✓ GET	/api/customers/{customerId}	Read a single customer
✎ PUT	/api/customers	Update an existing customer
DELETE	/api/customers/{customerId}	Delete an existing customer



Application Interaction

```
{
   "id" : 1,
   "firstName" : "Daniel",
   "lastName" : "Vega",
   "email" : "daniel.vega@luv2code.com"
}
```

```
{
   "id" : 1,
   "firstName" : "Daniel",
   "lastName" : "Vega",
   "email" : "daniel.vega@luv2code.com"
}
```



Update Customer

File: CustomerRestController.java

```
@RestController
@RequestMapping("/api")
public class CustomerRestController {
  ...

  // add mapping for PUT /customers  - update existing customer

  @PutMapping("/customers")
  public Customer updateCustomer(@RequestBody Customer theCustomer) {

    customerService.saveCustomer(theCustomer);

    return theCustomer;
  }

}
```

```
{
    "id" : 1,
    "firstName" : "Daniel",
    "lastName" : "Vega",
    "email" : "daniel.vega@luv2code.com"
}
```

Since customer ID is set, DAO will UPDATE customer in the database