Create a new Data Rest Project as per below details given below.

Project Name: Application

Classname: Customer

Attribute Names for Customer class: customerid , firstName, lastName, email, address

Create Repository class for the Customer Entity to map the following APIs with the query methods

HTTP Methods	routes	Action
POST	/customers	Create a new Customer
GET	/customers	Read all customers
GET	/customers/{customerid}	Read a single Customer
DELETE	/customers/{customerid}	Delete a single Customer
PUT	/customers/{customerid}	Update a single Customer

Code-

Customer.java (Entity)

```java
package com.ProductRestApi.model;

import java.time.Instant;

@Entity
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull(message = "First name must not be null")
    @Size(min = 3, message = "First name must contain at least 3 characters")
    private String firstName;

    @NotNull(message = "Last name must not be null")
    @Size(min = 3, message = "Last name must contain at least 3 characters")
    private String lastName;

    @NotNull(message = "Email must not be null")
    @Email(message = "Email should be valid")
    private String email;

    @NotNull(message = "Password must not be null")
    @Pattern(regexp = "^(?=.*[a-zA-Z])(?=.*\\d)[A-Za-z\\d]{6,20}$",
    message = "Password must be alphanumeric and 6-20 characters long")
    private String password;

    @NotNull(message = "Confirm password must not be null")
    @Pattern(regexp = "^(?=.*[a-zA-Z])(?=.*\\d)[A-Za-z\\d]{6,20}$",
    message = "Password must be alphanumeric and 6-20 characters long")
    //@Transient // Exclude from persistence since it won't be stored in the database
    private String confirmPassword;
```

```java
    @NotNull(message = "Mobile number must not be null")
    @NotEmpty(message = "Mobile number must not be empty")
    @Pattern(regexp = "\\d{10}", message = "Mobile number must be exactly 10 digits")
    private String mobileNo;

    @Min(value = 10000, message = "Salary must be at least 10,000")
    @Max(value = 50000, message = "Salary must not exceed 50,000")
    private double salary;

//    @CreatedDate
//    private Instant createdAt;  // Automatically set the creation timestamp
//
//    @LastModifiedDate
//    private Instant modifiedAt;  // Automatically set the modification timestamp
//
    @Column(name = "created_at", updatable = false)
    private Instant createdAt;

    @Column(name = "modified_at")
    private Instant modifiedAt;

    @NotNull(message = "Address must not be null")
    private String address;  // Added address field

    @PrePersist
    protected void onCreate() {
        createdAt = Instant.now();
    }

    @PreUpdate
    protected void onUpdate() {
        modifiedAt = Instant.now();
    }
```

```java
    // Getters and setters

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
```

```
121     public String getPassword() {
122         return password;
123     }
124
125     public void setPassword(String password) {
126         this.password = password;
127     }
128
129     public String getConfirmPassword() {
130         return confirmPassword;
131     }
132
133     public void setConfirmPassword(String confirmPassword) {
134         this.confirmPassword = confirmPassword;
135     }
136
137     public String getAddress() {
138         return address;
139     }
140
141     public void setAddress(String address) {
142         this.address = address;
143     }
144
145     public String getMobileNo() {
146         return mobileNo;
147     }
148
149     public void setMobileNo(String mobileNo) {
150         this.mobileNo = mobileNo;
151     }
152
```

```
153     public double getSalary() {
154         return salary;
155     }
156
157     public void setSalary(double salary) {
158         this.salary = salary;
159     }
160
161     public Instant getCreatedAt() {
162         return createdAt;
163     }
164
165     public void setCreatedAt(Instant createdAt) {
166         this.createdAt = createdAt;
167     }
168
169     public Instant getModifiedAt() {
170         return modifiedAt;
171     }
172
173     public void setModifiedAt(Instant modifiedAt) {
174         this.modifiedAt = modifiedAt;
175     }
176
177 }
178
```

CustomerController.java

```java
package com.ProductRestApi.controller;

import java.util.List;

@RestController
@RequestMapping("/api/customers")
public class CustomerController {

    @Autowired
    private CustomerService customerService;

    @GetMapping
    public List<Customer> getAllCustomers() {
        return customerService.getAllCustomers();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Customer> getCustomerById(@PathVariable Long id) {
        return customerService.getCustomerById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<Customer> addCustomer(@RequestBody Customer customer) {
        return ResponseEntity.ok(customerService.addCustomer(customer));
    }

    @PutMapping("/{id}")
    public ResponseEntity<Customer> updateCustomer(@PathVariable Long id, @RequestBody Customer customer) {
        return ResponseEntity.ok(customerService.updateCustomer(id, customer));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<String> deleteCustomer(@PathVariable Long id) {
        customerService.deleteCustomer(id);
        return ResponseEntity.ok("Customer deleted successfully!");
    }

    // Custom query endpoints
    @GetMapping("/search/firstName/{firstName}")
    public List<Customer> findByFirstName(@PathVariable String firstName) {
        return customerService.findByFirstName(firstName);
    }

    @GetMapping("/search/lastName/{lastName}")
    public List<Customer> findByLastName(@PathVariable String lastName) {
        return customerService.findByLastName(lastName);
    }

    @GetMapping("/search/email/{email}")
    public ResponseEntity<Customer> findByEmail(@PathVariable String email) {
        return ResponseEntity.ok(customerService.findByEmail(email));
    }

    @GetMapping("/search/salaryRange")
    public List<Customer> findBySalaryRange(@RequestParam double minSalary, @RequestParam double maxSalary) {
        return customerService.findBySalaryRange(minSalary, maxSalary);
    }

    @GetMapping("/search/salaryGreaterThan/{salary}")
    public List<Customer> findBySalaryGreaterThan(@PathVariable double salary) {
        return customerService.findBySalaryGreaterThan(salary);
    }

    @GetMapping("/search/nameStartingWith/{prefix}")
    public List<Customer> findByFirstNameStartingWith(@PathVariable String prefix) {
        return customerService.findByFirstNameStartingWith(prefix);
    }

    @GetMapping("/search/emailDomain")
    public List<Customer> findByEmailDomain(@RequestParam String domain) {
        return customerService.findByEmailDomain(domain);
    }

}
```

CustomerService.java

```java
package com.ProductRestApi.service;

import java.util.List;

@Service
public class CustomerService {

    @Autowired
        private CustomerRepository customerRepository;

        public List<Customer> getAllCustomers() {
            return customerRepository.findAll();
        }

        public Optional<Customer> getCustomerById(Long id) {
            return customerRepository.findById(id);
        }

        public Customer addCustomer(Customer customer) {
            if (!customer.getPassword().equals(customer.getConfirmPassword())) {
                throw new IllegalArgumentException("Passwords do not match");
            }
            return customerRepository.save(customer);
        }

        public Customer updateCustomer(Long id, Customer customer) {
            if (!customerRepository.existsById(id)) {
                throw new IllegalArgumentException("Customer not found");
            }
            customer.setId(id);
            return customerRepository.save(customer);
        }

        public void deleteCustomer(Long id) {
            if (!customerRepository.existsById(id)) {
                throw new IllegalArgumentException("Customer not found");
            }
            customerRepository.deleteById(id);
```

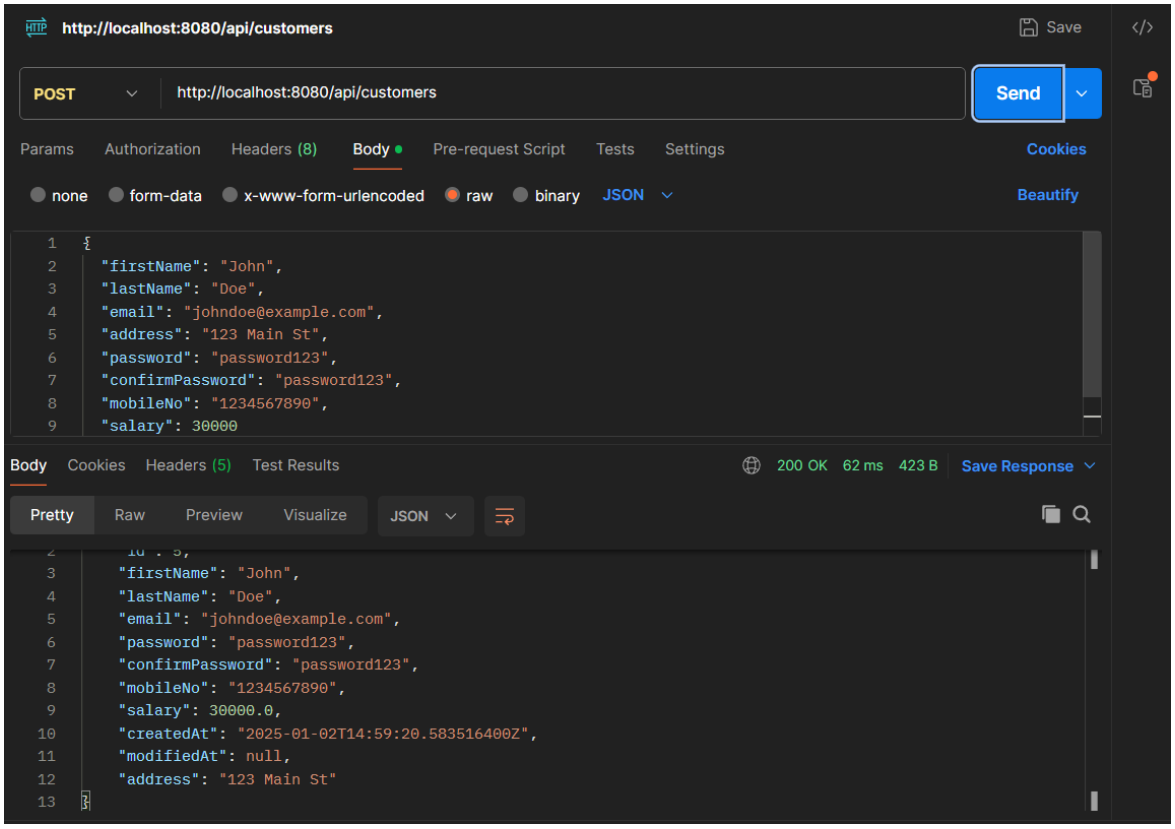
```
        }

        // Query methods
        public List<Customer> findByFirstName(String firstName) {
            return customerRepository.findByFirstName(firstName);
        }

        public List<Customer> findByLastName(String lastName) {
            return customerRepository.findByLastName(lastName);
        }

        public Customer findByEmail(String email) {
            return customerRepository.findByEmail(email);
        }

        public List<Customer> findBySalaryRange(double minSalary, double maxSalary) {
            return customerRepository.findBySalaryBetween(minSalary, maxSalary);
        }

        public List<Customer> findBySalaryGreaterThan(double salary) {
            return customerRepository.findBySalaryGreaterThan(salary);
        }

        public List<Customer> findByFirstNameStartingWith(String prefix) {
            return customerRepository.findByFirstNameStartingWith(prefix);
        }

        public List<Customer> findByEmailDomain(String domain) {
            return customerRepository.findByEmailDomain(domain);
        }

}

```

CustomerRepository

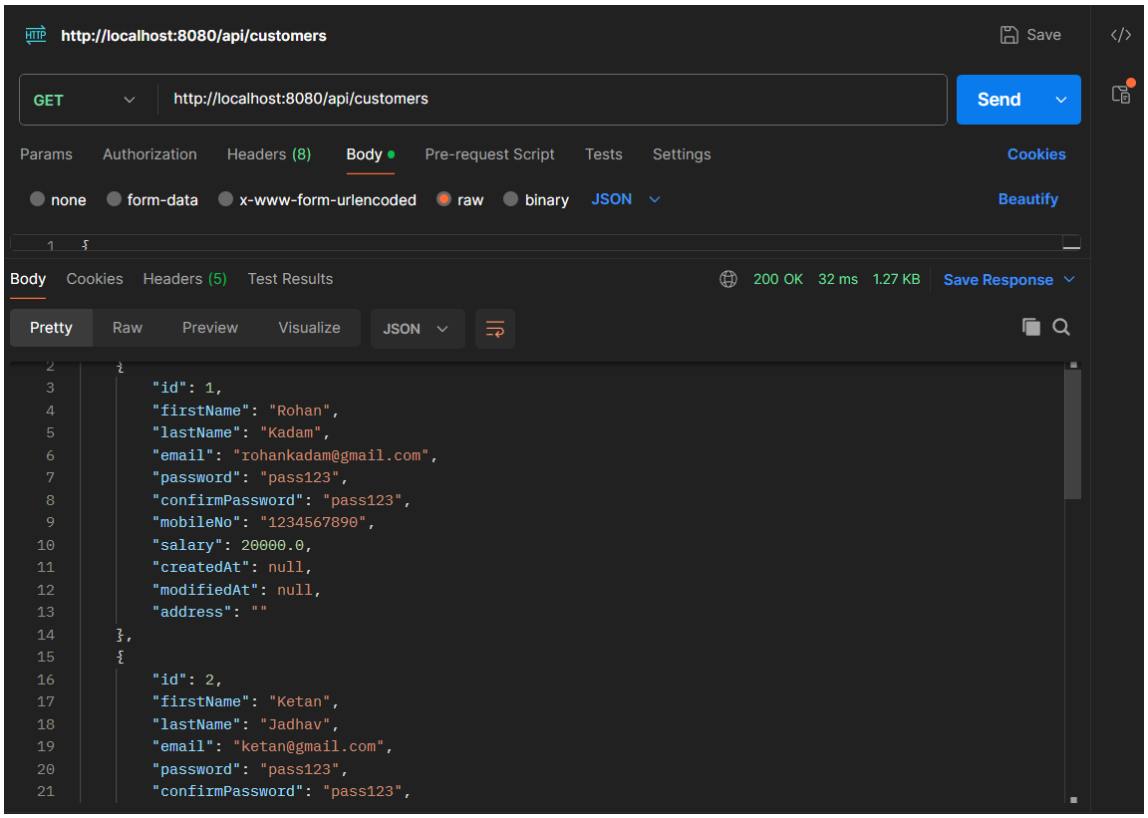
```
package com.ProductRestApi.repository;

import java.util.List;

public interface CustomerRepository extends JpaRepository<Customer, Long>{

    // Find Customers by First Name
    List<Customer> findByFirstName(String firstName);

    // Find Customers by Last Name
    List<Customer> findByLastName(String lastName);

    // Find Customers by Email
    Customer findByEmail(String email);

    // Find Customers by Salary Range
    List<Customer> findBySalaryBetween(double minSalary, double maxSalary);

    // Find Customers with Salary Greater Than a Specific Value
    List<Customer> findBySalaryGreaterThan(double salary);

    // Find Customers by Name Starting With
    List<Customer> findByFirstNameStartingWith(String prefix);

    // Find Customers by Email Domain
    @Query("SELECT c FROM Customer c WHERE c.email LIKE %:domain")
    List<Customer> findByEmailDomain(@Param("domain") String domain);

}

```

Output

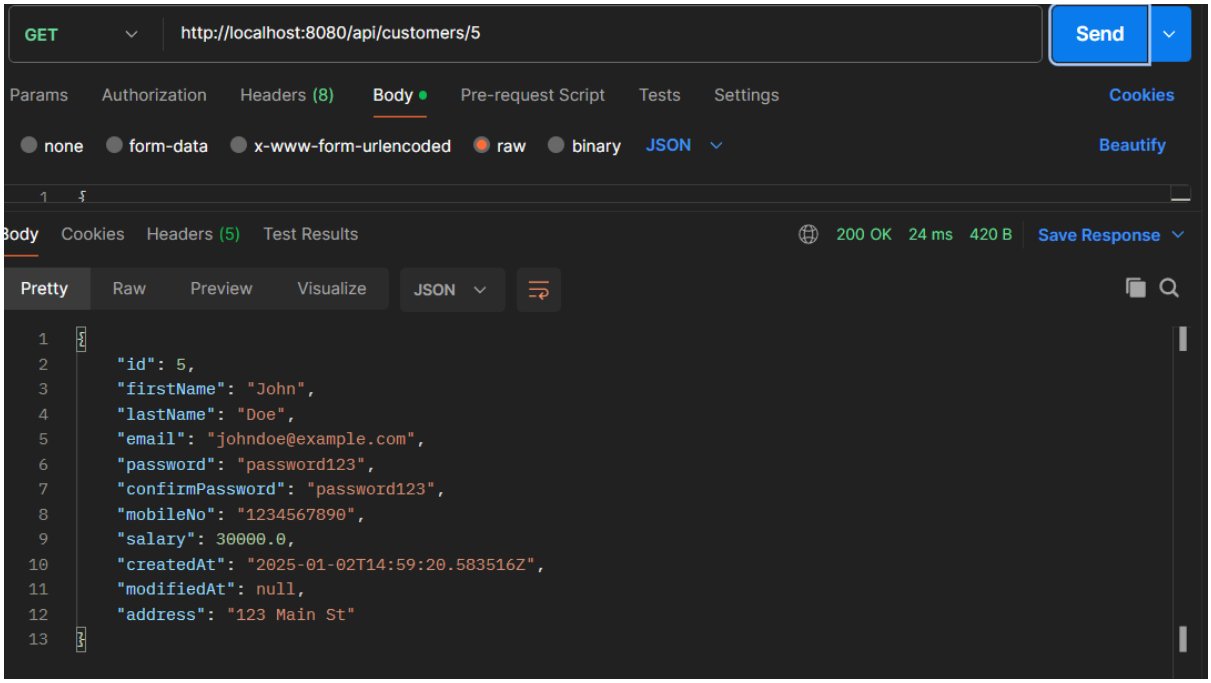
Add the Customer



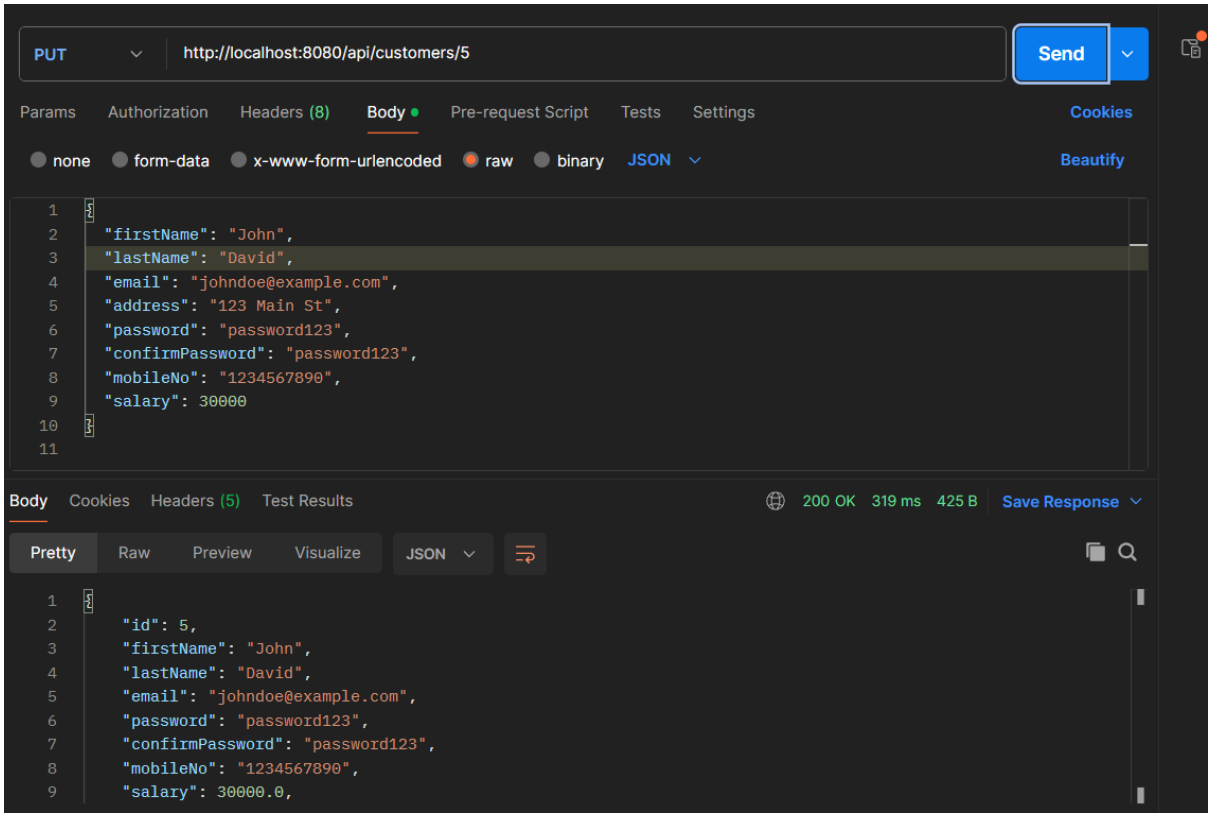
Get the All Customer



Get the single customer by id



Update the single customer data by id



Delete the Data of Customer by id