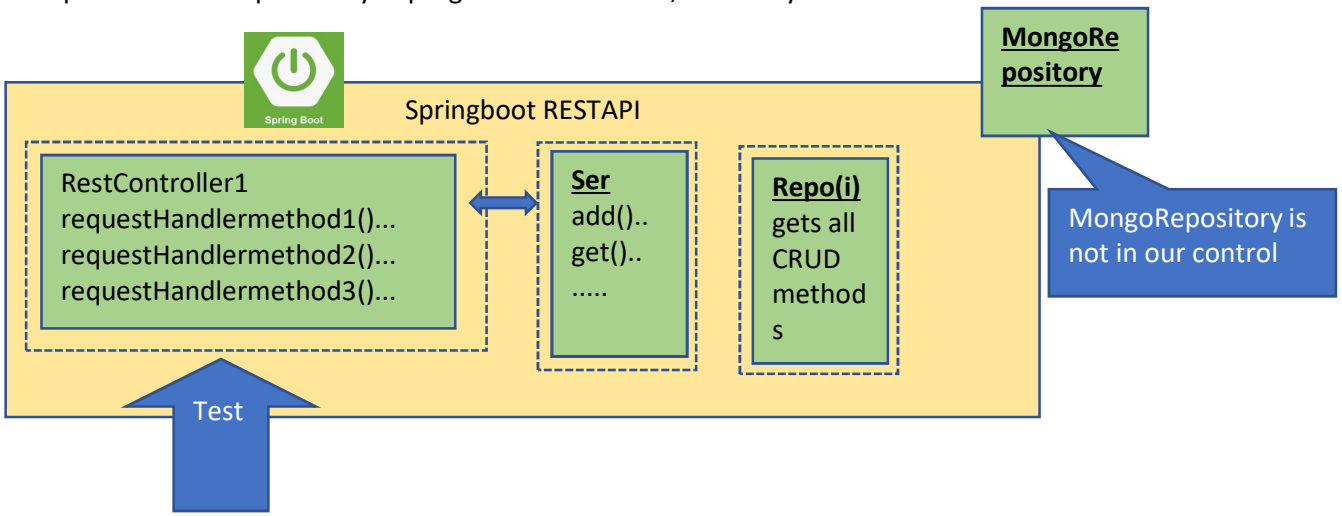
If both Dependent and Dependency in programmers controll, then only mocker environment can be created for test



Mocito

creating mocked environment for dependency layer
so, dependent layer can work with mocked dependency layer without effecting original resources

Controller -> Service

Dependent Dependency
@InjectMocks @Mock

Are controller request handling methods same as service methods or repo methods ?

No

service methods / repo methods

return type method name (arg list)

here, methods can be called as normal method

ex
customerService.addCustomer(customer);
customerRepository.findAll();

But, controller methods execute based on URL request
these request handler methods are not executed as usual

Are we calling controller method as this?
customerController.addCustomer();
No

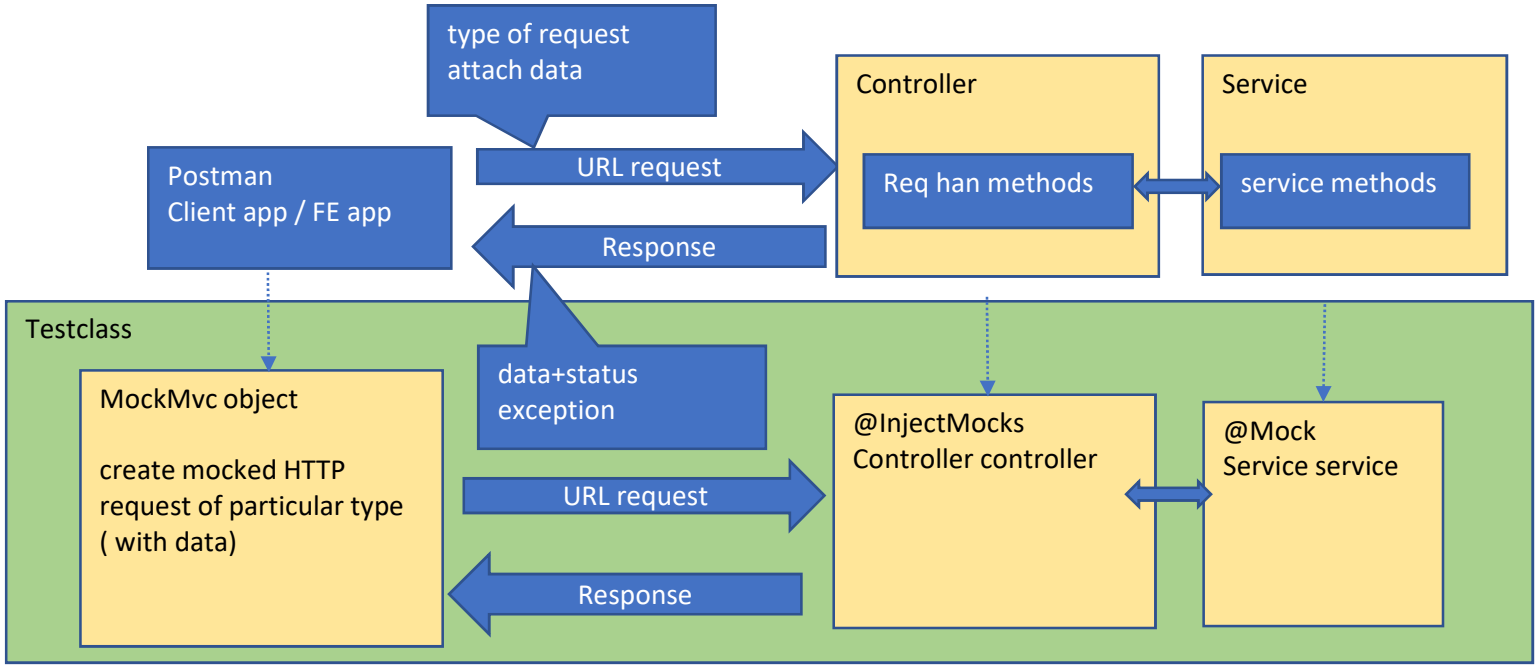
The below request handler methods execute when repective URL request reached to controller

```
18      // GET
19      // http://localhost:9999/customerapp/v1/customer
20      @GetMapping("/customer")
21      public ResponseEntity<?> getAllCustomers(){
22          return new ResponseEntity<>(customerService.getAllCustomers(), HttpStatus.OK);
23      }
24
25      // POST
26      // http://localhost:9999/customerapp/v1/customer
27      @PostMapping("/customer")
28      public ResponseEntity<?> addCustomer(@RequestBody Customer customer) throws CustomerAlreadyExistingException{
29          try{
30              return new ResponseEntity<>(customerService.addCustomer(customer), HttpStatus.OK);
31          }
32          catch(CustomerAlreadyExistingException ex){
33              throw new CustomerAlreadyExistingException();
34          }
35      }
```

MockMvc

FE application / Postman

http://localhost:9999/customerapp/v1/customer GET	CustomerController getAllCustomers()
http://localhost:9999/customerapp/v1/customer POST	CustomerController addCustomer()



Step 1 Create Test class with proper annotation

create @Mock Service object

create @IntectMocks Controller object

create autowired MockMvc object

create Customer, Address objects

```
14  @ExtendWith(MockitoExtension.class)
15  public class CustomerControllerTest {
16
17      @Autowired
18      private MockMvc mockMvc;
19
20      @Mock
21      private CustomerService customerService;
22
23      @InjectMocks
24      private CustomerController customerController;
25
26      private Customer customer;
27      private Address address;
```

Step 2 Define setup and clean method

```
33      @BeforeEach
34      public void init(){
35          address = new Address( doorNo: "123", street: "s1", area: "area1", city: "city1");
36          customer = new Customer( customerId: "C1", name: "krishna", email: "nomail.com", mobile: "1234",address);
37          mockMvc = MockMvcBuilders.standaloneSetup(customerController).build();
38      }
39
40      @AfterEach
41      public void clean(){
42          address=null;
43          customer=null;
44      }
```

Step 3 write test case method

```
74      private static String convertToJson(final Object obj){
75          String result="";
76          try{
77              ObjectMapper mapper = new ObjectMapper();
78              result = mapper.writeValueAsString(obj);
79          }
80          catch(Exception ex){
81              ex.printStackTrace();
82          }
83          return result;
84      }
```

```
55      @Test
56      public void addCustomerSuccess() throws Exception {
57          when(customerService.addCustomer(customer)).thenReturn(customer);
58
59          // need to send HTTP request as
60          // POST, http://localhost:9999/customerapp/v1/customer
61          // with customer object as attached along with request
62                      // controller addCustomer() method called
63          // get response as 'OK', display response
64          mockMvc.perform(
65                  post( urlTemplate: "/customerapp/v1/customer")
66                          .contentType(MediaType.APPLICATION_JSON)
67                          .content(convertToJson(customer)))
68                          // controller addCustomer() executes
69                  .andExpect(status().isOk()).andDo(MockMvcResultHandlers.print());
70
71          verify(customerService,times( wantedNumberOfInvocations: 1)).addCustomer(customer);
72      }
```

```java
@Test
public void addCustomerFailure() throws Exception {
    when(customerService.addCustomer(customer)).thenThrow(CustomerAlreadyExistingException.class);
    mockMvc.perform(
            post( urlTemplate: "/customerapp/v1/customer")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(convertToJson(customer)))
                    // controller.addCustomer();
            .andExpect(status().isConflict()).andDo(MockMvcResultHandlers.print());
    verify(customerService,times( wantedNumberOfInvocations: 1)).addCustomer(customer);
}
```