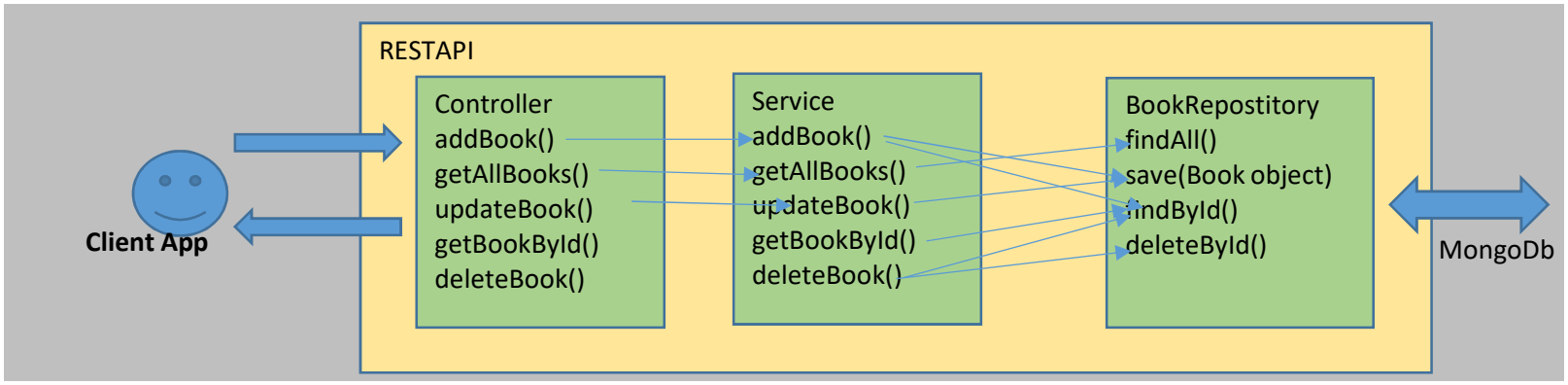
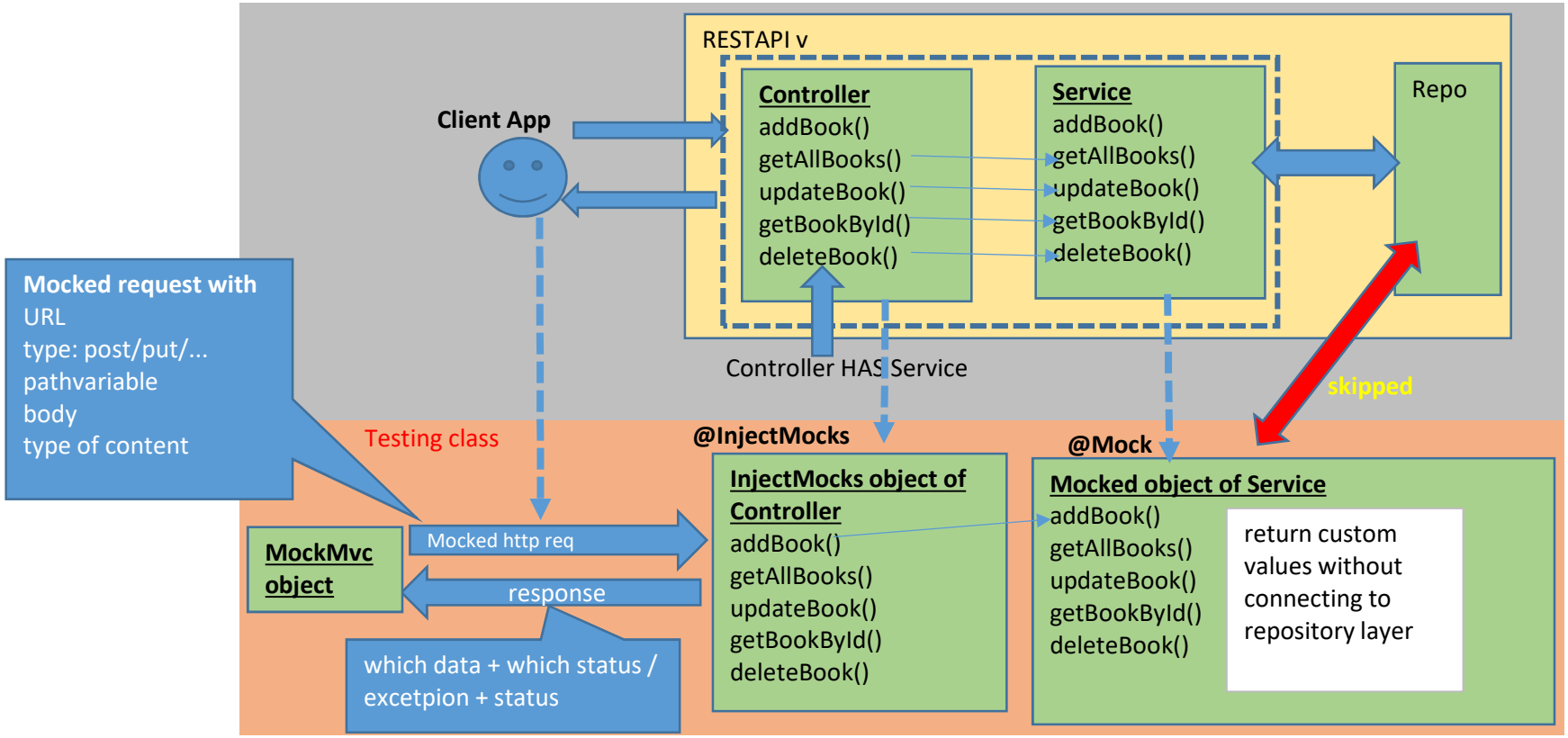
How controller request handler methods are getting invoked
by client app / acting client app

Request handler methods in controller are invoked by HTTP Request



Controller is dependent

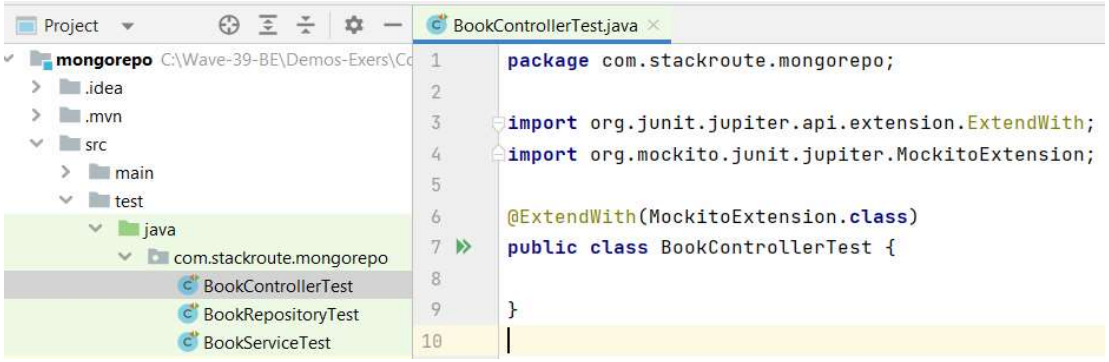
Service is dependency



Steps to write test cases for controller layer

Step 1 Make sure springboot appliation ready with complete flow
copy sprint3 (mongocrud)

Step 2 create new test class in test package



Step 3 create @Mock object of service
create @InjectMocks object of controller
create @Autowired object of MockMvc
create Author and Book objects

```
14  @ExtendWith(MockitoExtension.class)
15  public class BookControllerTest {
16
17      @Autowired
18      private MockMvc mockMvc;
19
20      @Mock
21      private BookService bookService;
22
23      @InjectMocks
24      private BookController bookController;
25
26      private Book book;
27      private Author author;
28  }
```

Step 4 define setup, clean

```
33      @BeforeEach
34      public void setup(){
35          author = new Author( authorName: "Bg Sw", address: "Chennai", age: 35);
36          book = new Book( bookId: "BK0001", name: "Spirit of C", subject: "C",author, price: 345, stock: 56);
37          book1 = new Book( bookId: "BK0002", name: "abcd", subject: "xyz",
38                  new Author( authorName: "xyz", address: "Chennai", age: 34),
39                   price: 345, stock: 43);
40          mockMvc = MockMvcBuilders.standaloneSetup(bookController).build();
41      }
42
43      @AfterEach
44      public void clean(){
45          book=null; book1=null; author=null;
46      }
```

Step 5 write test cases

Test 1 test addBook() request (Success)

```
17      /* POST
18      http://localhost:65500/addBook
19      */
20      @PostMapping("/addBook")
21      public ResponseEntity<?> addBook(@RequestBody Book book) throws BookAlreadyExistingException {
22          // return new ResponseEntity<>(bookService.addBook(book), HttpStatus.OK);
23          // in failure case : if needed to log info somewhere, to send alert mail or . . . .
24          try{
25              return new ResponseEntity<>(bookService.addBook(book), HttpStatus.OK);
26          }
27          catch(BookAlreadyExistingException ex){
28              // log /email . . .
29              throw new BookAlreadyExistingException();
30          }
31      }
```

controller method

MockRequest

URL: http://localhost:65500/addBook,
Type: post, contentType: Application/Json,
data : JSON type of Book object

when(service.addBook(book)).then return book object
..
this request will invoke controller.addBook()
check reponse
verify()

```java
@Test
public void testControllerAddBookSuccess() throws Exception {
    when(bookService.addBook(book)).thenReturn(book);
    // need to create post request with URL /addBook
    // attach book data as JSON object
    mockMvc.perform(
            post( urlTemplate: "/addBook")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(convertToJson(book))) // raises req, controller method gets invoked
                    // ABOVE LINES OF CODE execute controller.addBook()--calls--> service.addBook()
            .andExpect(status().isOk()).andDo(MockMvcResultHandlers.print()); // response
    verify(bookService,times( wantedNumberOfInvocations: 1)).addBook(book);
}
```

create http request with BASE_URL/addBook
method as post, content type as Application_JSON,
content as JSON type of book object
This request invokes controller.addBook()

Method to convert Java object into JSON object

```java
// method to convert Java object into JSON object
1 usage
public static String convertToJson(final Object object){
    String result="";
    ObjectMapper mapper= new ObjectMapper();
    try {
        result = mapper.writeValueAsString(object);
    } catch (JsonProcessingException e) {
        throw new RuntimeException(e);
    }
    return result;
}
```

Test 2 addBook failure

```
73      @Test
74      public void testControllerAddBookFailure() throws Exception {
75          when(bookService.addBook(book)).thenThrow(BookAlreadyExistingException.class);
76          mockMvc.perform(
77                  post( urlTemplate: "/addBook")
78                          .contentType(MediaType.APPLICATION_JSON)
79                          .content(convertToJson(book)))
80                  .andExpect(status().isConflict()).andDo(MockMvcResultHandlers.print());
81          verify(bookService,times( wantedNumberOfInvocations: 1)).addBook(book);
82      }
```

Test 3 delete Book success

```
75      @Test
76      public void testControllerAddBookFailure() throws Exception {
77          when(bookService.addBook(book)).thenThrow(BookAlreadyExistingException.class);
78          mockMvc.perform(
79                  post( urlTemplate: "/addBook")
80                          .contentType(MediaType.APPLICATION_JSON)
81                          .content(convertToJson(book)))
82                  .andExpect(status().isConflict()).andDo(MockMvcResultHandlers.print());
83          verify(bookService,times( wantedNumberOfInvocations: 1)).addBook(book);
84      }
```

Test 4 delete Book failure

```
97      @Test
98      public void testControllerDeleteBookFailure() throws Exception {
99          when(bookService.deleteBook( bkid: "BK0001")).thenThrow(BookNotFoundException.class);
100         mockMvc.perform(
101                 delete( urlTemplate: "/delete-book/BK0001"))
102                 .andExpect(status().isNotFound()).andDo(MockMvcResultHandlers.print());
103         verify(bookService,times( wantedNumberOfInvocations: 1)).deleteBook( bkid: "BK0001");
104     }
```