luv2code

Testing Spring MVC Web Controllers

Problem

- How can we test Spring MVC Web Controllers?
- How can we create HTTP requests and send to the controller?
- How can we verify HTTP response?
  - status code
  - view name
  - model attributes

Spring Testing Support

- Mock object support for web, REST APIs etc ...
- For testing controllers, you can use `MockMvc`
- Provides Spring MVC processing of request / response
- There is no need to run a server (embedded or external)

Development Process

1. Add annotation `@AutoConfigureMockMvc`
2. Inject the `MockMvc`
3. Perform web requests
4. Define expectations
5. Assert results

GradebookController.java

```
@Controller
public class GradebookController {

  @RequestMapping(value = "/", method = RequestMethod.GET)
  public String getStudents(Model m) {
    return "index";
  }

}
```

GradebookControllerTest.java

```
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
...

@AutoConfigureMockMvc
@SpringBootTest
public class GradebookControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void getStudentsHttpRequest () throws Exception {

        MvcResult mvcResult = mockMvc.perform(MockMvcRequestBuilders.get("/"))
                .andExpect(status().isOk()).andReturn();

        ModelAndView mav = mvcResult.getModelAndView();

        ModelAndViewAssert.assertViewName(mav, "index");

    }

}
```

Step 1: Autoconfigure

Step 2: Inject the MockMvc

Step3: Perform web requests

Step 4: Define expectations

Step 5: Assert results

Can also assert model attributes.
Retrieve model attribute objects for fine-grained asserts

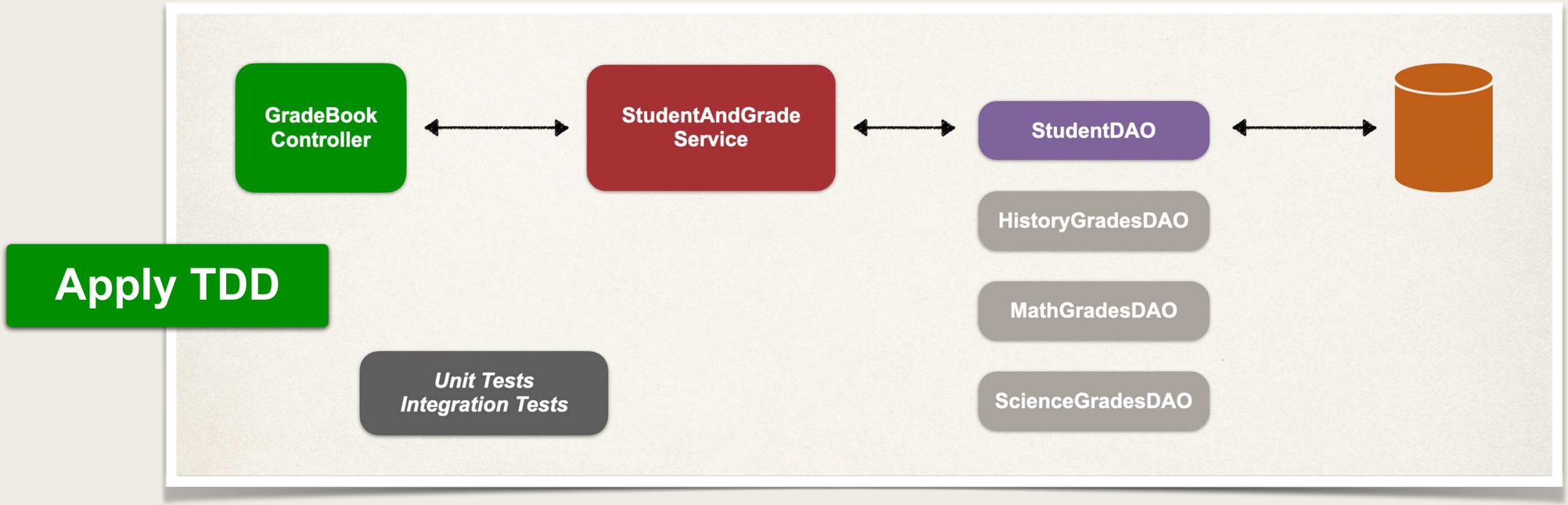
GradebookController.java

```
@Controller
public class GradebookController {

  @RequestMapping(value = "/", method = RequestMethod.GET)
  public String getStudents(Model m) {
    return "index";
  }

}
```

Test - Create Student

Test Case: Create a student in the database

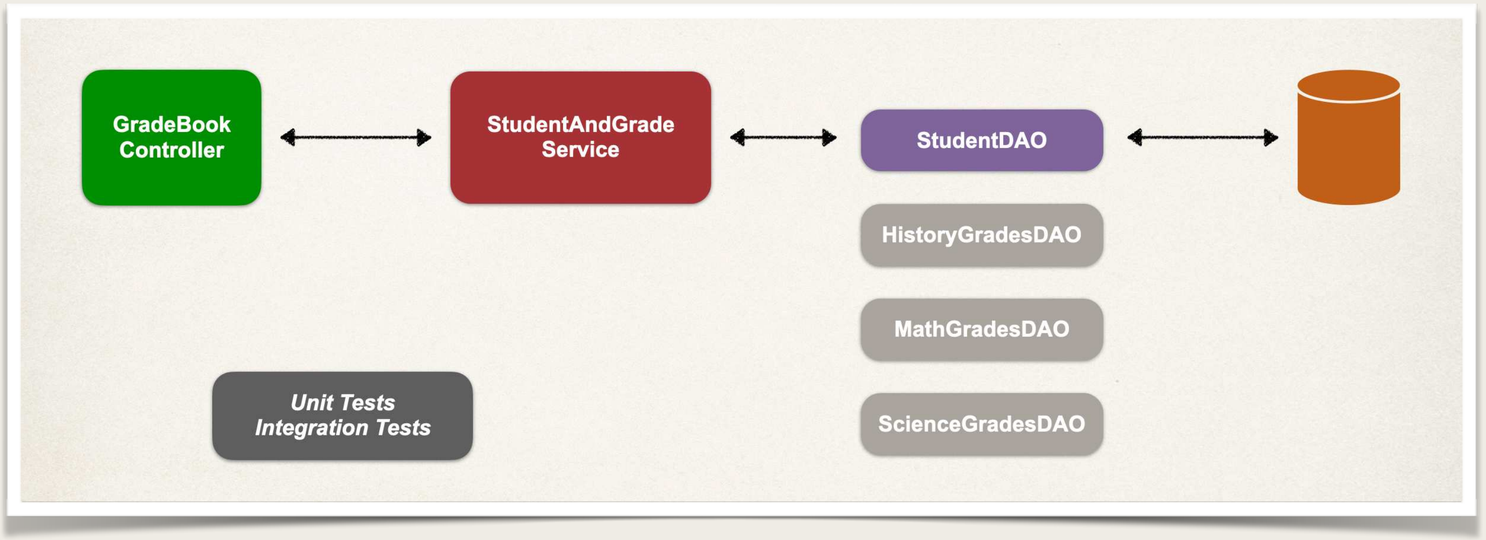
- Send a POST request to the controller
- Verify results by accessing data using the DAO



Updates for Gradebook UI

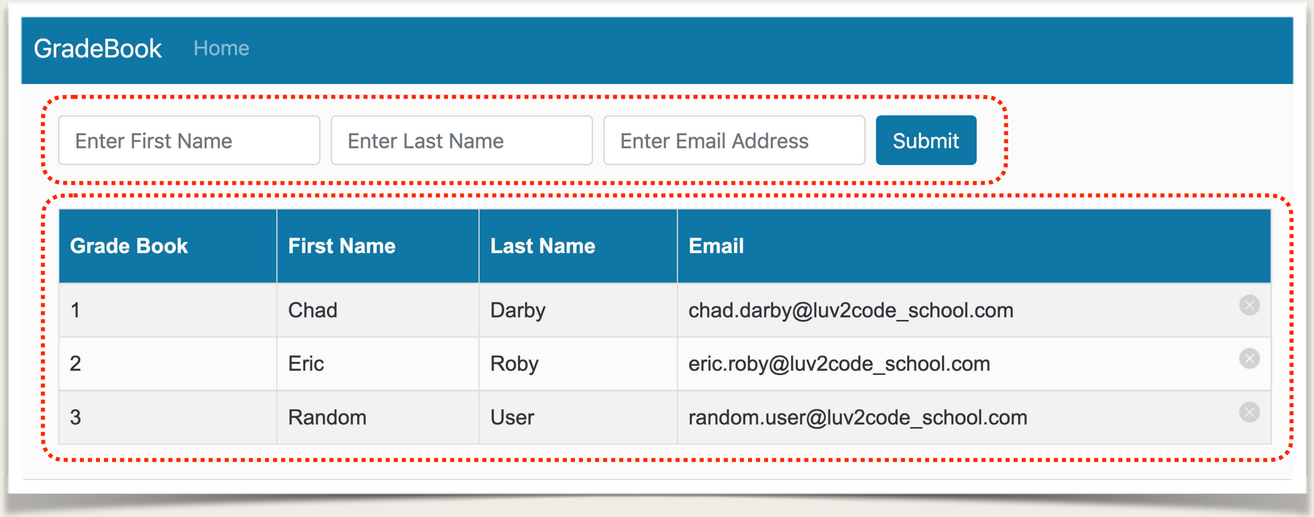
Check Point

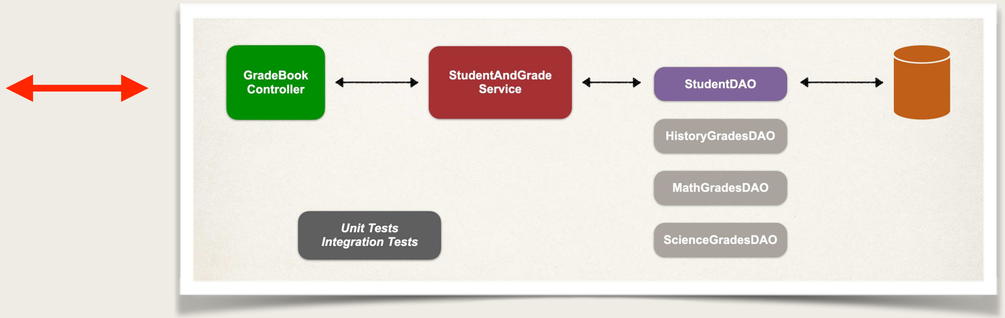
- At the moment, we have developed tests for
  - Creating a student
  - Getting a list of students



Work To Do

- The current UI has hard coded HTML ... doesn't really do anything
- To Do
  - Update index.html to have form submit data to GradeBookController
  - Update index.html to display list of students using a for loop over the "students" model attribute





Delete Student

To Do: Delete Student

- Apply TDD
  - Create a failing test
  - Add code to GradeBookController to delete student ... make the test pass
  - Add code to GradeBookController to check for error page ... make the test pass