**Tutorial 4 - Menggunakan Database dan Melakukan Debugging dalam Project Spring Boot**

**Latihan Menambahkan Delete**

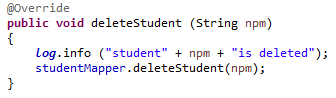
1. Pada tutorial sebelumnya Anda sudah menambahkan method delete. Sekarang implementasikan method tersebut menggunakan database.
2. Pada viewall.html tambahkan



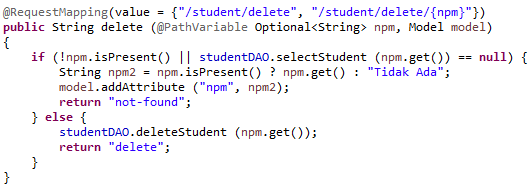
1. Tambahkan method deleteStudent yang ada di class StudentMapper



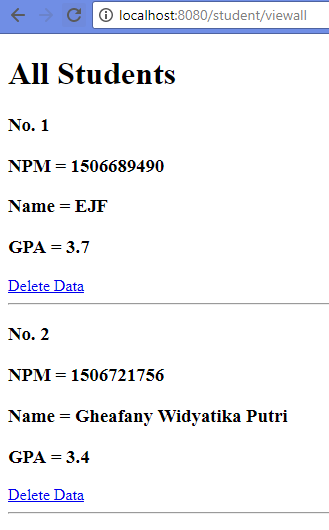
1. Lengkapi method deleteStudent yang ada di class StudentServiceDatabase



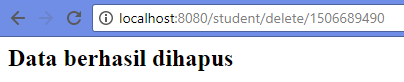
1. Lengkapi method delete pada class StudentController



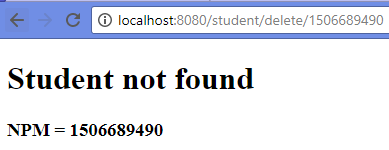
**Contoh tampilan View All setelah di Insert**



**Contoh tampilan jika dilakukan delete NPM 123**



**Contoh tampilan jika dilakukan delete NPM 123 yang kedua kalinya**



**Contoh tampilan jika dilakukan delete yang tidak memasukan NPM di URL**



**Penjelasan method Delete:**

Ketika URL /student/delete atau /student/delete/{npm} dipanggil, maka akan dilakukan pengecekan, jika NPM tidak terdapat pada URL maka akan menampilkan halaman notfound.html yang akan mem-*print* “Student Not Found ; NPM = Tidak ada”, sama halnya dengan NPM yang dimasukan tidak ada di dalam *database,* maka akan menampilkan halaman notfound.html yang akan mem-*print* “Student Not Found ; NPM = (sesuai dengan NPM yang dimasukan dalam URL” sedangkan jika NPM ada maka akan dicari objek *student* yang memiliki NPM tersebut, jika ada maka objek student tersebut akan dihapus dengan mengeksekusi Query method *delete* dan akan menampilkan halaman delete.html

**Latihan Menambahkan Update**

Selama ini Anda masih menggunakan method GET untuk melakukan masukan. Pada update kali ini, Anda akan mencoba menggunakan method POST.

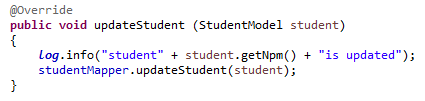
1. Tambahkan method updateStudent pada class StudentMapper



1. Tambahkan method updateStudent pada interface StudentService



1. Tambahkan implementasi method updateStudent pada class StudentServiceDatabase. Jangan lupa tambahkan logging pada method ini.



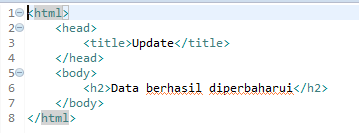
1. Tambahkan link Update Data pada viewall.html



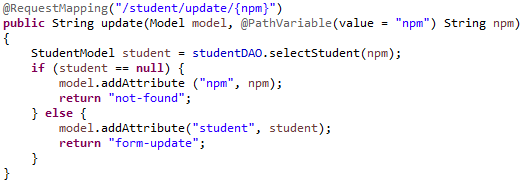
1. Copy view form-add.html menjadi form-update.html



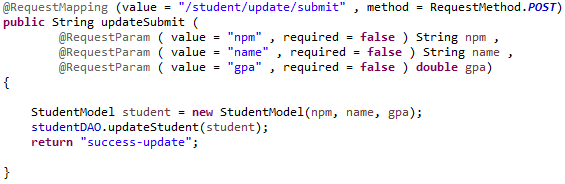
1. Copy view success-add.html menjadi success-update.html



1. Tambahkan method update pada class StudentController



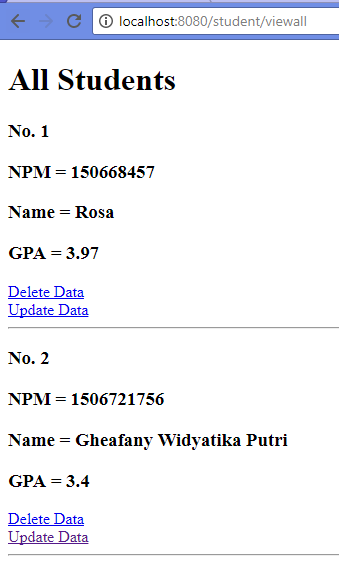
1. Tambahkan method updateSubmit pada class StudentController



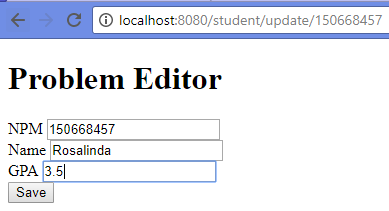
**Penjelasan Method Update**:

Ketika URL /student/update/{npm} dipanggil, method *update* akan mengecek apakah terdapat objek *student* dengan NPM tersebut di dalam *database*, jika tidak ada maka akan mengembalikan notfound.html, namun jika NPM tersebut ada di dalam *database* maka form-update.html untuk mulai mengisikan perubahan kemudian data-data *student* akan dikirim melalui model. Apabila tombol *submit* telah ditekan maka akan masuk kedalam method updateSubmit yang kemudian akan return success-update untuk menampilkan halaman success-update.html

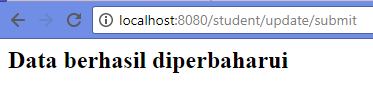
**Contoh tampilan View All setelah di Insert**



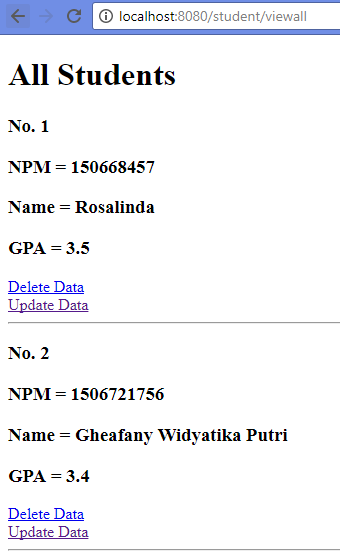
**Contoh tampilan Form Update**



**Contoh tampilan setelah Success Update**

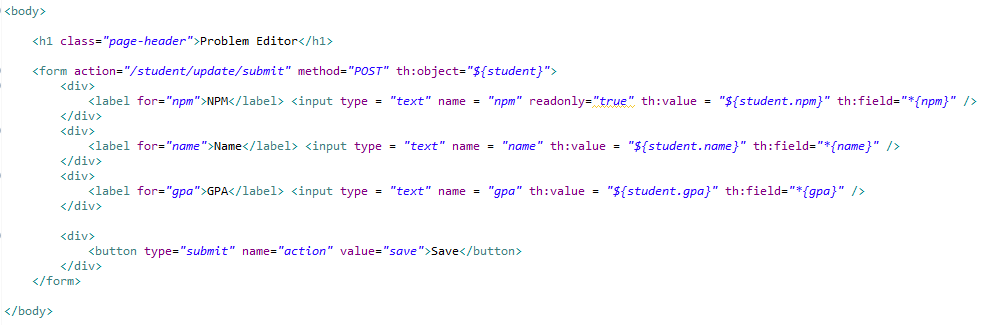


**Contoh tampilan View All setelah dilakukan update**

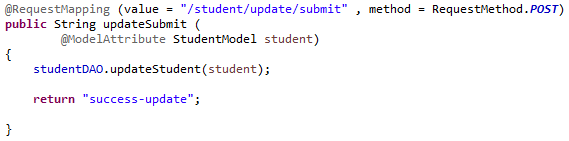


**Latihan Menggunakan Object Sebagai Parameter**

1. Pada tutorial di atas Anda masih menggunakan RequestParam untuk menghandle form submit. Sehingga ada banyak parameter pada method Anda. Bayangkan jika Anda memiliki form yang memiliki banyak field, maka parameternya akan sangat banyak dan tidak rapih.
2. SpringBoot dan Thymeleaf memungkinkan agar method updateSubmit menerima parameter berupa model StudentModel . Metode ini lebih disarankan dibandingkan menggunakan RequestParam.
3. A. Menambahkan th:object=”${student}” pada tag <form> di view kemudian menambahkan th:field=”\*{[nama\_field]}” pada setiap input



B. Ubah method updateSubmit pada StudentController yang hanya menerima parameter berupa StudentModel



**Penjelasan**:

Pada method ini, mengubah penggunaan RequestParam agar menjadi object, pada kasus ini yaitu StudentModel, pada form-update.html memberikan th:object=”${student}” pada tag <form>, th:field=”\*{[nama\_field]} pada setiap input, mengubah param updateSubmit dengan StudentController sehingga menerima parameter berupa StudentModel menggunakan @ModelAttribute.

**Pertanyaan**

Beberapa pertanyaan yang perlu dijawab:

1. Jika menggunakan Object sebagai parameter pada form POST, bagaimana caranya melakukan validasi input yang optional dan input yang required? Apakah validasi diperlukan? Asumsikan input pada form Anda tidak menggunakan attribute required sehingga butuh validasi di backend.

**Jawab:**

Mengecek apakah property dari objek null atau tidak, dapat di handle pada controllernya, dimana gpa tidak boleh kosong, hal tersebut disebabkan karena apabila gpa kosong maka textbox akan mengembalikan tipe string, sedangkan gpa merupakan double sehingga akan menimbulkan error. mengecek apakah student.npm tidak ada isi / null sehingga input tidak bisa ditinggalkan.

1. Menurut Anda, mengapa form submit biasanya menggunakan POST method dibanding GET method? Apakah perlu penanganan berbeda di header atau body method di controller jika form di post dikirim menggunakan method berbeda?

**Jawab:**

Pada *form submit* akan melakukan perubahan resource pada server-side atau penambahan *resource* baru. Jika menggunakan method POST, paramater-parameter tidak akan tersimpan dalam *history* pada *browser* dan juga tidak dapat di*bookmark* sehingga menjadi lebih aman dari serangan orang-orang yang tidak bertanggung jawab serta aman dari *spamming* . Sedangkan jika kita menggunakan method GET, parameter-parameternya akan tersimpan dalam *history* pada *browser*. Hal ini bisa dengan mudah dimanfaatkan oleh orang-orang jahil dan tidak bertanggung jawab untuk menyerang aplikasi kita, selain itu mereka juga dapat melakukan spamming terhadap URL karena method GET juga dapat di*bookmark*, hal tersebut dapat menyebabkan *website* menjadi *down* akibat banyaknya *request* ke *server*. Penganan berbeda di header atau body method di controller diperlukan karena untuk setiap parameter method memiliki RequestMethod memiliki parameter yang berbeda (RequestParameter.GET dan RequestParameter.POST)

1. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?

**Jawab:**

Hal tersebut mungkin terjadi. Beberapa caranya antara lain melakukan pengecekan *request type* pada *method body controller* atau bisa juga dengan cara menambahkan method = { RequestMethod.GET, RequestMethod.POST } pada parameter @RequestMapping atau dengan menggunakan Class HttpServletRequest

**Hal yang saya pelajari dari tutorial ini**

Pada tutorial kali ini, saya belajar bagaimana cara mengkoneksikan spring framework dengan *database* dan belajar bagaimana cara merangkai aplikasi dengan prinsip OOP serta tak lupa dengan MVC. Ternyata kita juga dapat menggunakan objek sebagai parameter dalam menerima http request.