**LAPORAN PRAKTIKUM**

**Mission 4 - Enhancing Interactivity with DOM & JS**



**Disusun Oleh:**

**Dzakir Tsabit Asy Syafiq (241511071)**

**Jurusan Teknik Komputer dan Informatika**

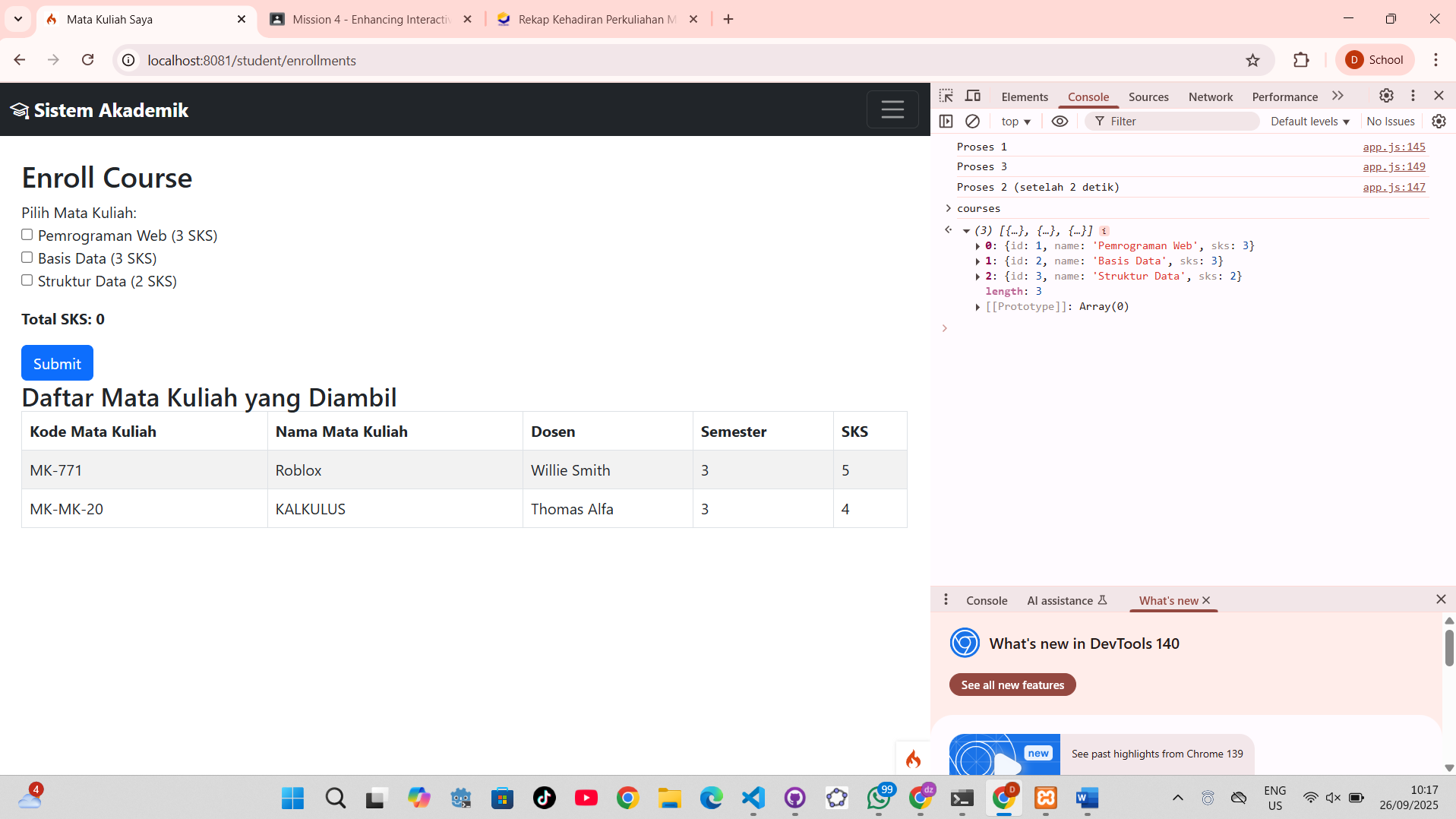
**Program Studi D-3 Teknik Informatika**

**Politeknik Negeri Bandung**

**23/09/2025**

Link Github : [*https://github.com/dzhax499/website-CRUD-ci4/tree/main/4-Misson-CI-Composer*](https://github.com/dzhax499/website-CRUD-ci4/tree/main/4-Misson-CI-Composer)

**1. Scope, Array, Object**



* **Scope**

Dibagian scope, variable enrollments di deklarasikan ke dalam scope global di dalam file /js/app.js

* **Array**

Array adalah yang menampung lebih dari 1 variable yang ditandai dengan kurung []

* **Object**

kumpulan pasangan *key-value* (kunci-nilai) yang digunakan untuk menyimpan data yang lebih kompleks. Objek ditandai dengan kurung kurawal {}. Contoh: { id: 1, name: 'Pemrograman Web', sks: 3 }

* **Array of Objects**

Ini adalah struktur data yang paling umum digunakan. Membuat sebuah *array* yang setiap elemennya adalah sebuah *object*. Ini sangat ideal untuk menyimpan daftar data yang terstruktur, seperti daftar mahasiswa atau mata kuliah

**Penjelasan Kode**

di dalam app.js saya, code nya seperti ini :  
 // Data Mahasiswa

const students = [

  { id: 1, name: 'Dzakir Tsabit', nim: '241511071' },

  { id: 2, name: 'John Doe', nim: '241511072' }

];

// Data Mata Kuliah

const courses = [

  { id: 1, name: 'Pemrograman Web', sks: 3 },

  { id: 2, name: 'Basis Data', sks: 3 },

  { id: 3, name: 'Struktur Data', sks: 2 }

];

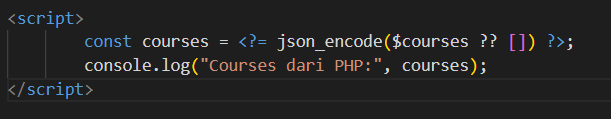
* Students

Sebuah array yang berisi dua objek. Setiap objek merepresentasikan satu mahasiswa dan memiliki properti id, name, dan nim

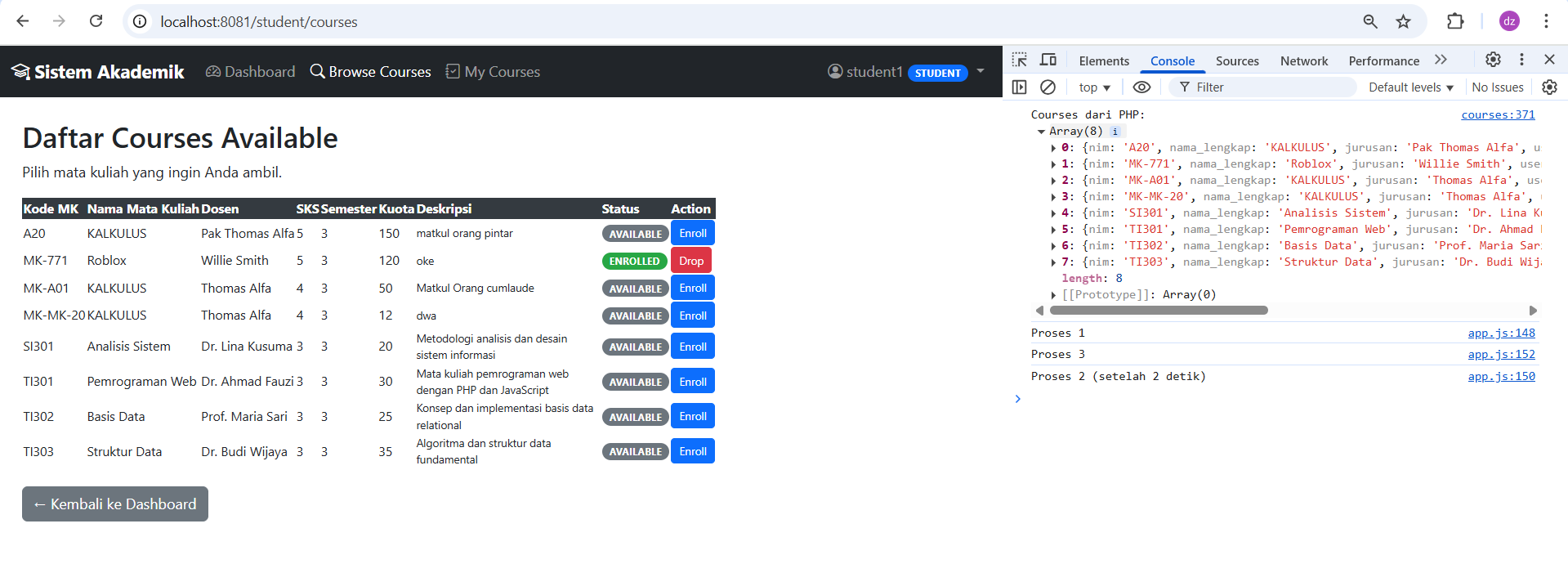
* Courses

Array yang berisi objek-objek mata kuliah, masing-masing dengan properti id, name, dan sks

Jika menggunakan database, maka cukup seperti ini code nya:



Ini hasil nya :



**2. DOM Selector dan Manipulation**

Sebelum kita menyelam ke dalam kode, mari kita bayangkan sejenak sebuah halaman web sebagai sebuah pohon keluarga. Ada "kakek buyut" (<html>), yang punya anak (<head> dan <body>), dan terus bercabang hingga ke "cucu-cicit" terkecil seperti tombol (<button>) atau sebuah paragraf (<p>). Struktur pohon inilah yang kita sebut sebagai **Document Object Model (DOM)**. Pada era web kuno, pohon ini kaku; sekali ditampilkan, ia tidak akan berubah kecuali seluruh halaman dimuat ulang.

Namun, zaman telah berubah. Pengguna modern menuntut pengalaman yang mulus dan instan. Di sinilah **DOM Manipulation** berperan sebagai sihirnya. Dengan JavaScript, kita tidak lagi hanya menjadi penonton; kita bisa menjadi "dalang" yang mengubah struktur pohon DOM ini sesuka hati. Kita bisa memerintahkan JavaScript untuk mencari (Selector) elemen spesifik—misalnya, sebuah <span> yang bertugas menampilkan total SKS—lalu memanipulasi (Manipulation) isinya secara langsung, tanpa perlu meminta *browser* untuk menggambar ulang seluruh halaman dari awal. Inilah esensi dari aplikasi web modern yang responsif.

Dalam proyek ini, sihir ini terlihat jelas. tidak memaksa mahasiswa untuk menghitung SKS secara manual atau menunggu halaman *refresh* hanya untuk melihat total SKS. Ini telah menggunakan JavaScript untuk "mengawasi" setiap *checkbox* dan secara dinamis mengubah teks di dalam elemen total SKS

**Penjelasan Konsep**

DOM (Document Object Model) adalah representasi struktur HTML dalam bentuk tree/pohon yang bisa dimanipulasi dengan JavaScript. Setiap elemen HTML (div, input, button, dll) adalah node dalam tree tersebut.

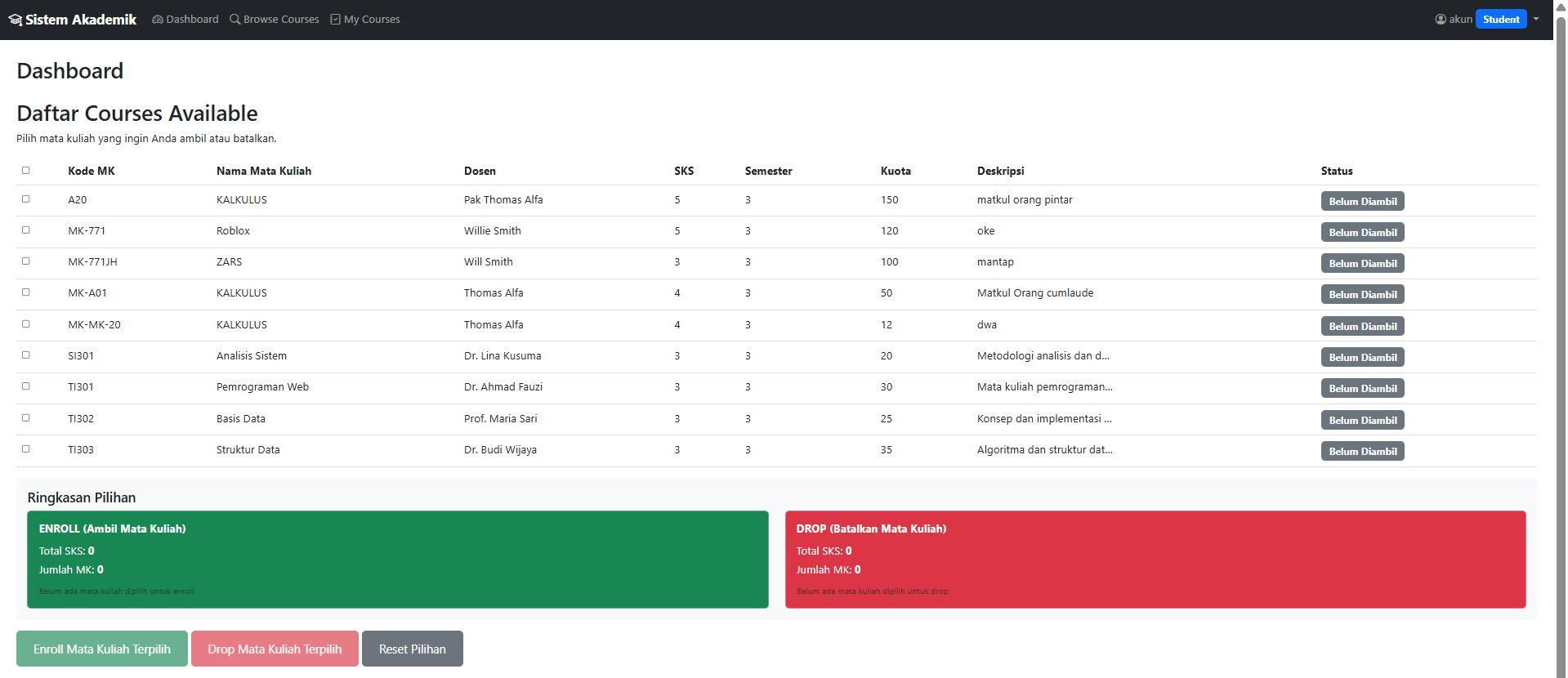
**DOM Selector** = cara mencari/mengakses elemen HTML **DOM Manipulation** = mengubah, menambah, atau menghapus elemen

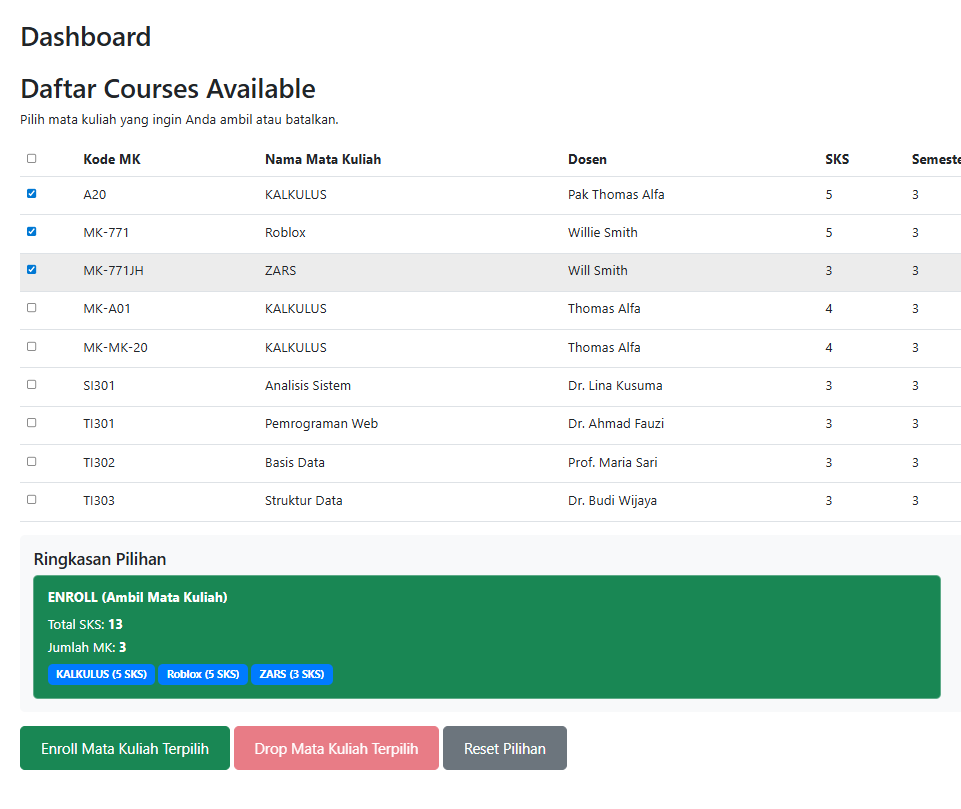
**Kaitan dengan Definition of Done (DoD):**

Implementasi ini secara langsung memenuhi salah satu syarat krusial dari DoD:

* Semua interaksi berjalan tanpa refresh halaman (JS DOM based)

Perubahan total SKS yang terjadi secara instan saat *checkbox* dicentang adalah bukti utama dari interaksi berbasis DOM yang berhasil dibangun





**Event Handling**

JavaScript berperan seperti sutradara di panggung teater. Panggungnya adalah halaman web, dan aktornya adalah elemen HTML seperti tombol dan form. **Event handling** adalah naskah yang memberi tahu aktor kapan harus beraksi. JavaScript menggunakan addEventListener untuk "mendengarkan" kejadian di halaman, seperti klik, perubahan checkbox, atau pengiriman form. Saat kejadian itu terjadi, JavaScript akan menjalankan fungsi tertentu, membuat halaman menjadi interaktif. Jadi, tanpa event handling, elemen di halaman cuma sekadar tampilan, tidak ada interaksi

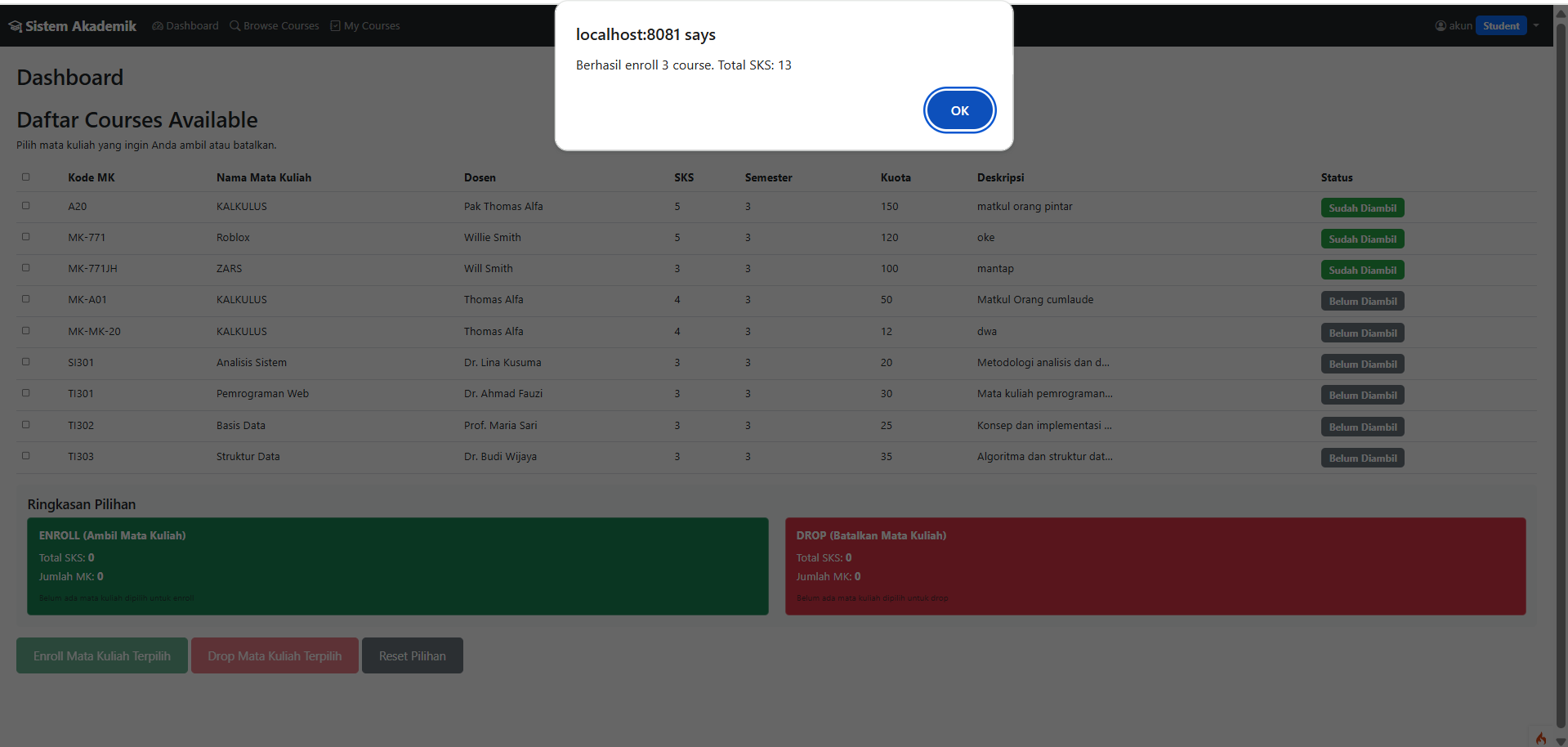
**Kaitan dengan Definition of Done (DoD):**

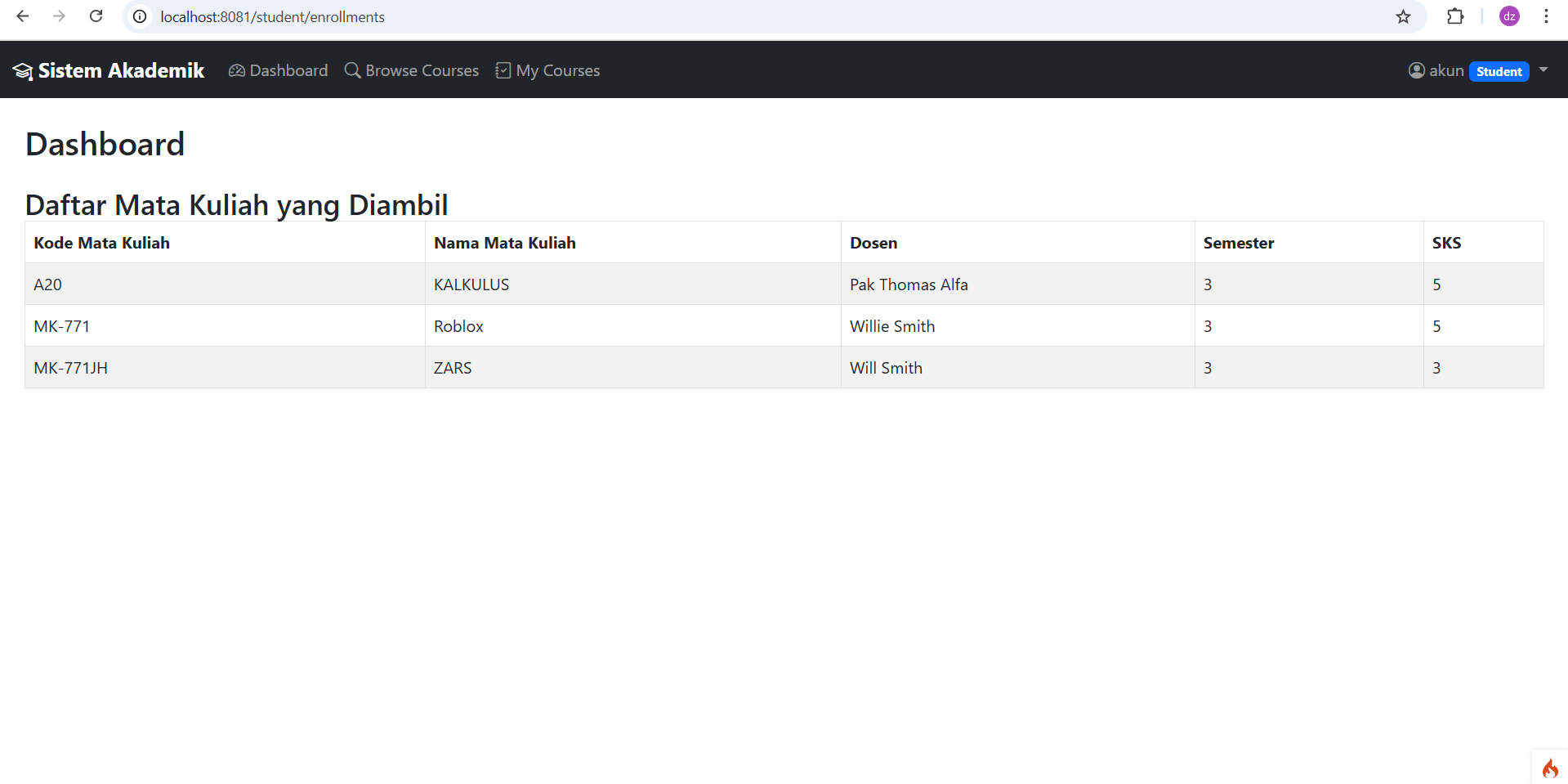
Penerapan Event Handling ini memenuhi beberapa poin DoD sekaligus:

* Sistem mendukung student enroll course dengan validasi (tidak bisa double enroll).
* Admin dapat create student & course dengan validasi input.

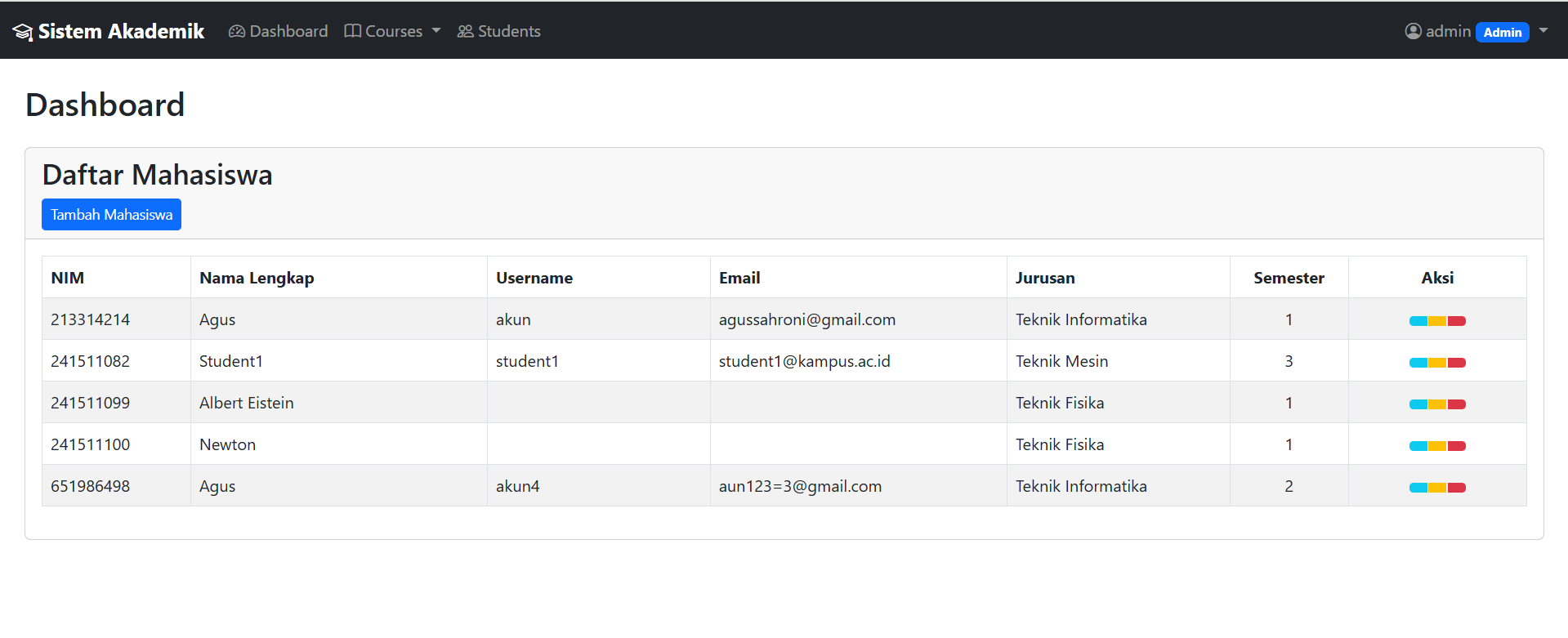
Meskipun validasi utama terjadi di *backend*, proses *enrollment* dan *create* itu sendiri diawali oleh sebuah *event* (klik tombol submit) yang ditangani

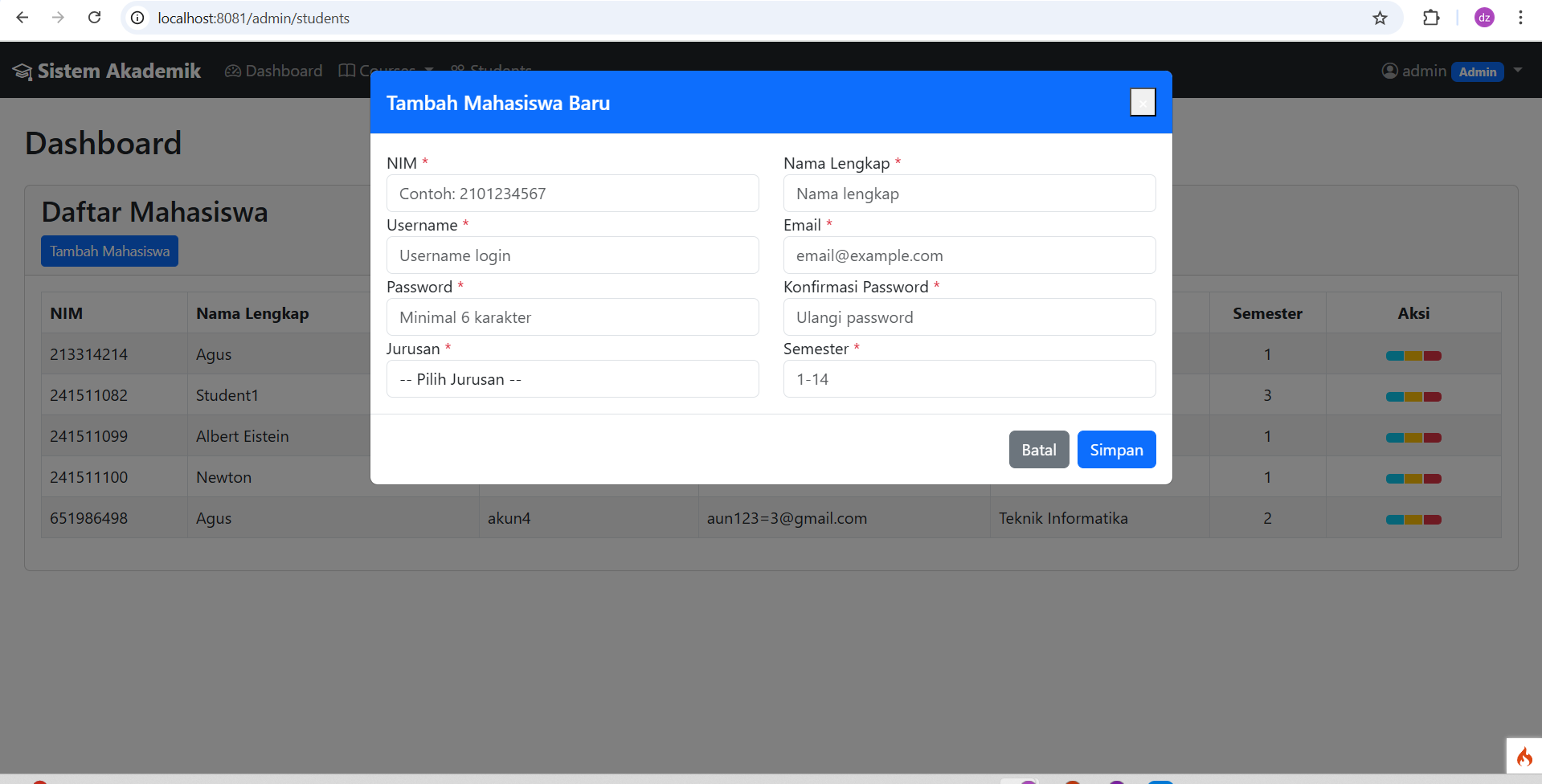
**Student view :**

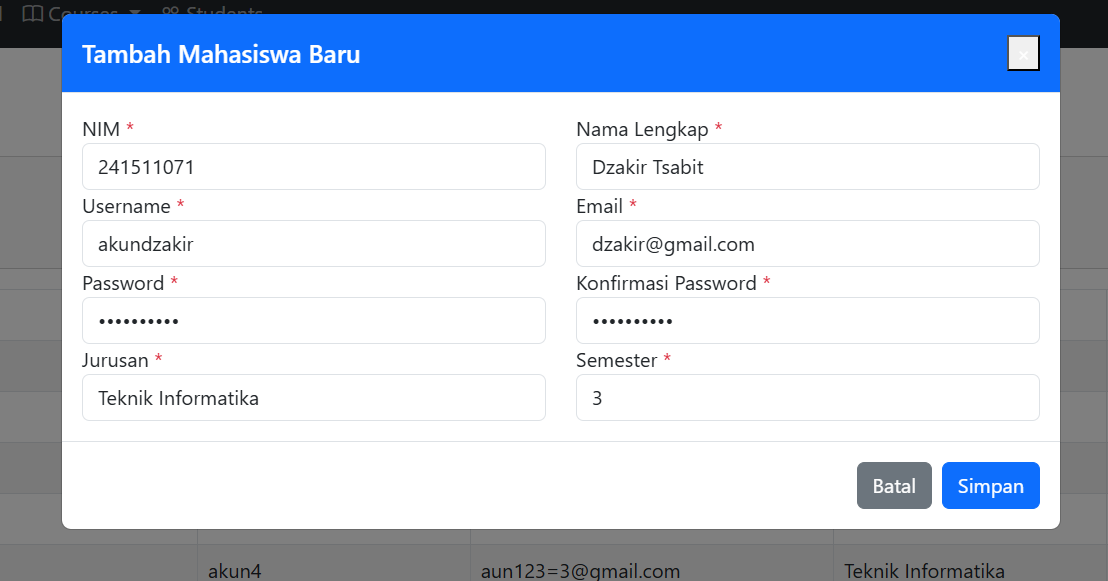


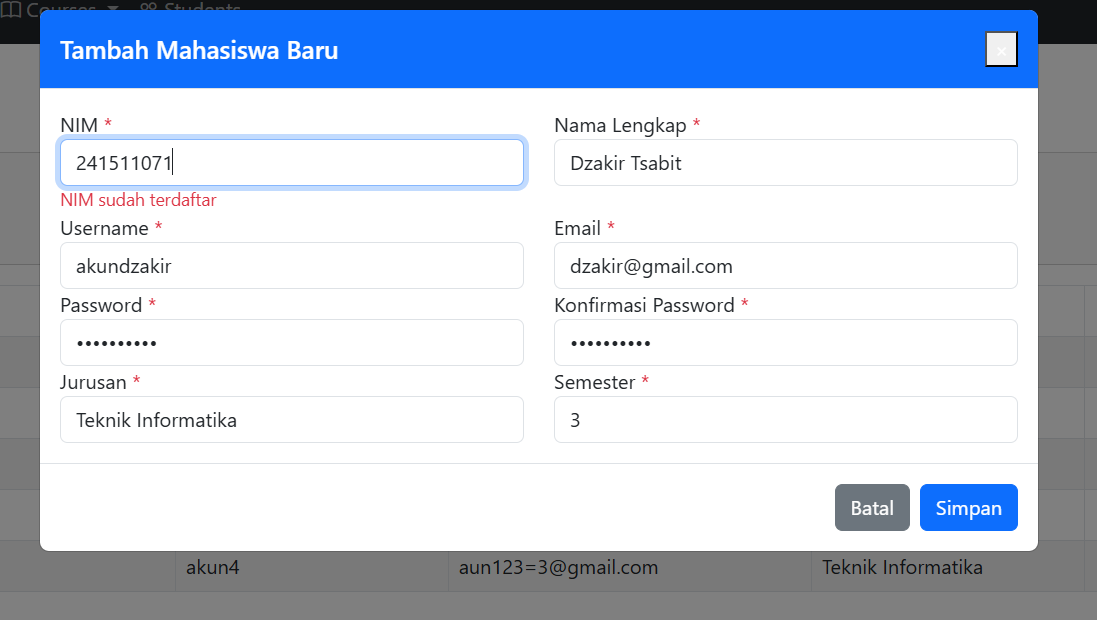


**Admin (view):**

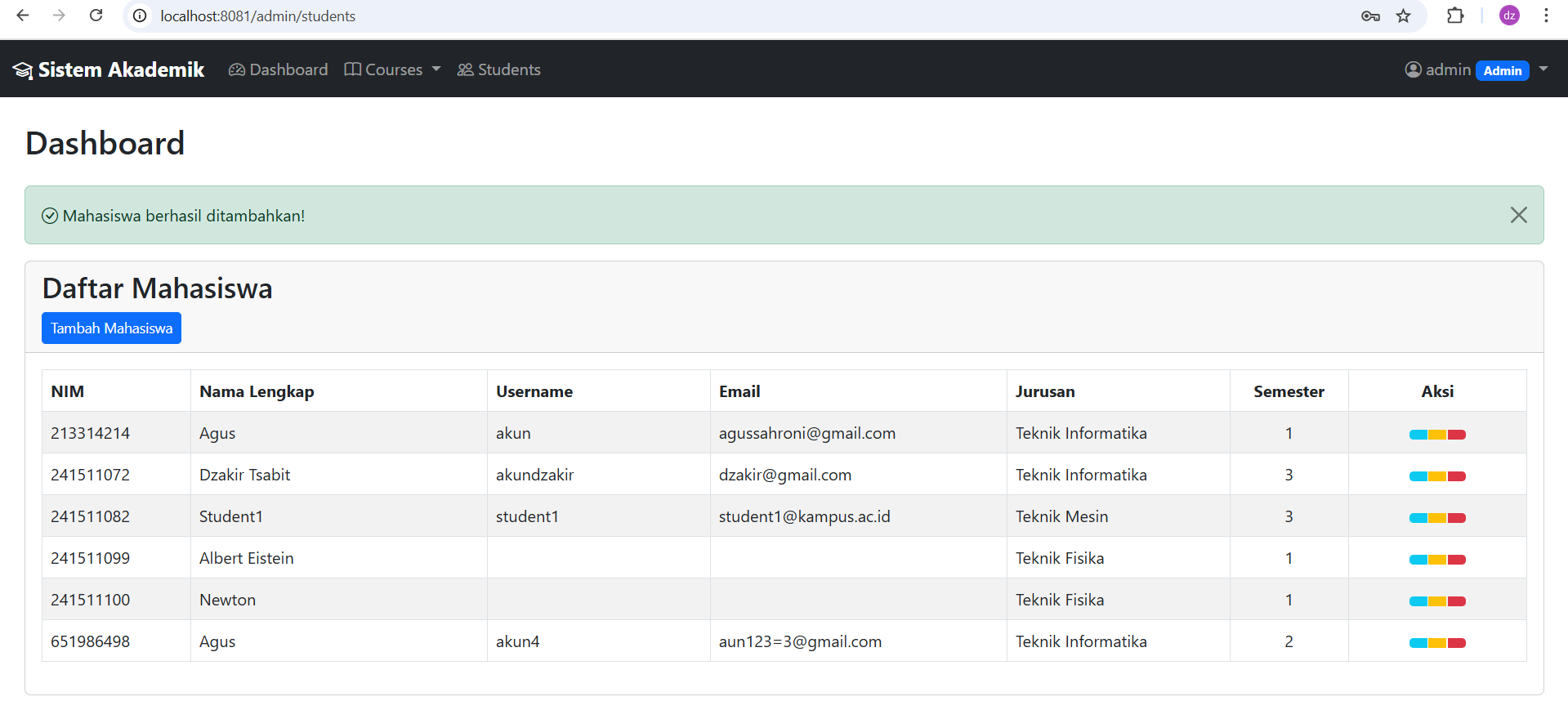
****

****

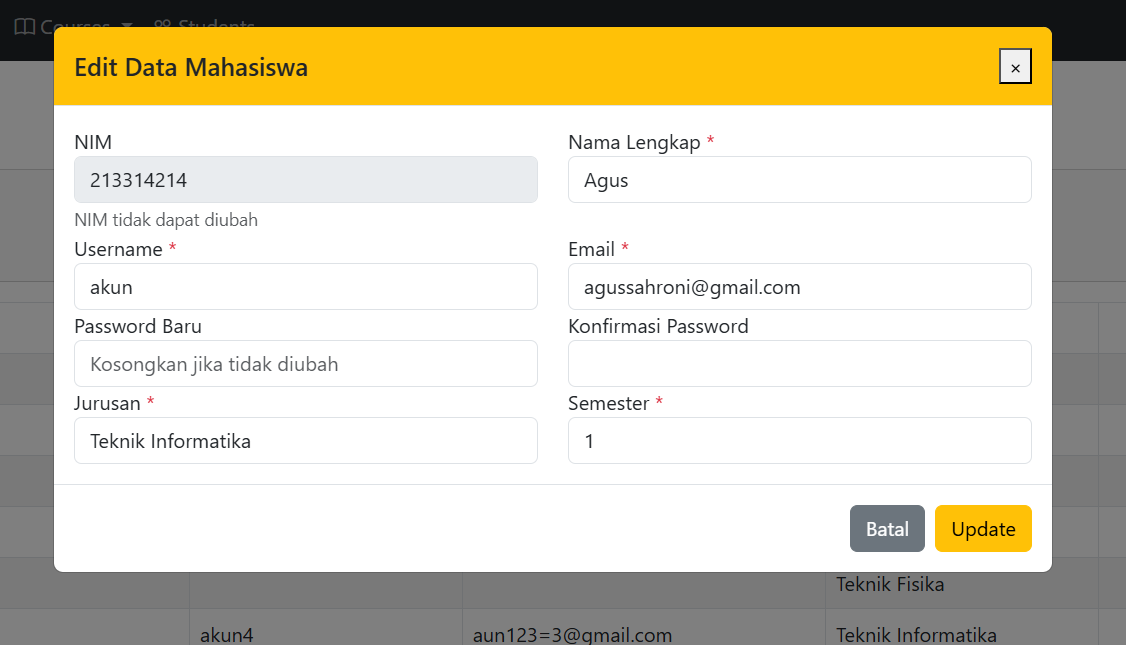
****

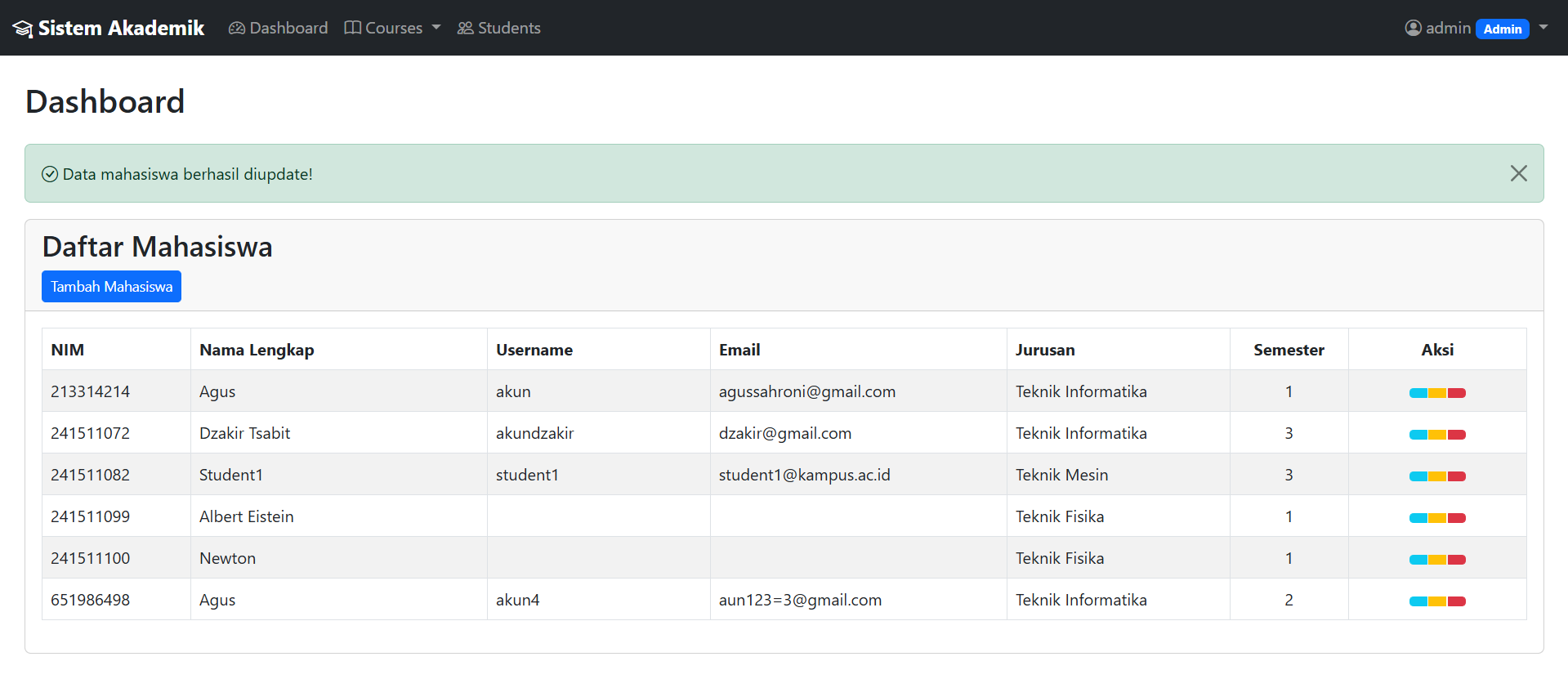
Jika sudah terdaftar akan seperti ini:  


Jika belum :

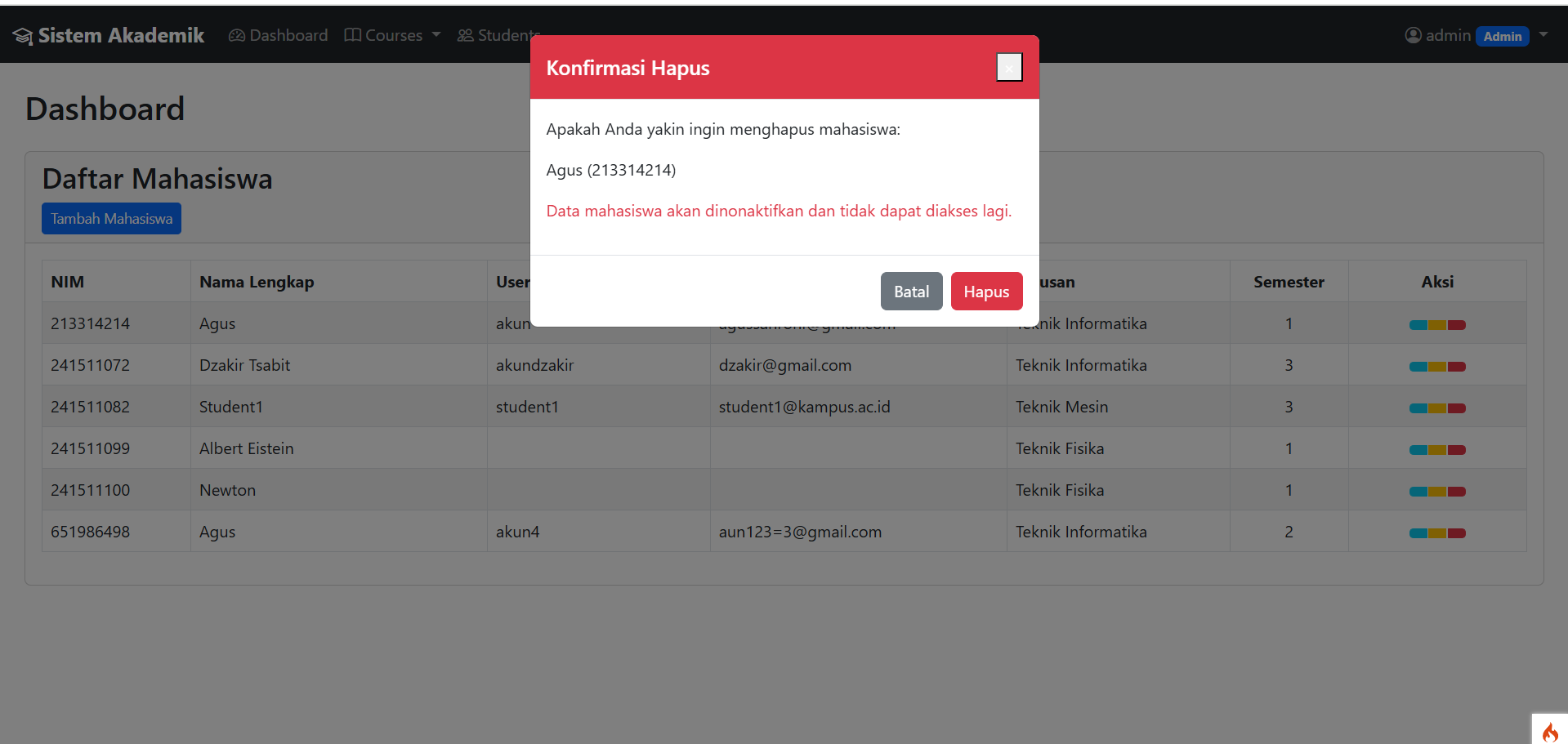


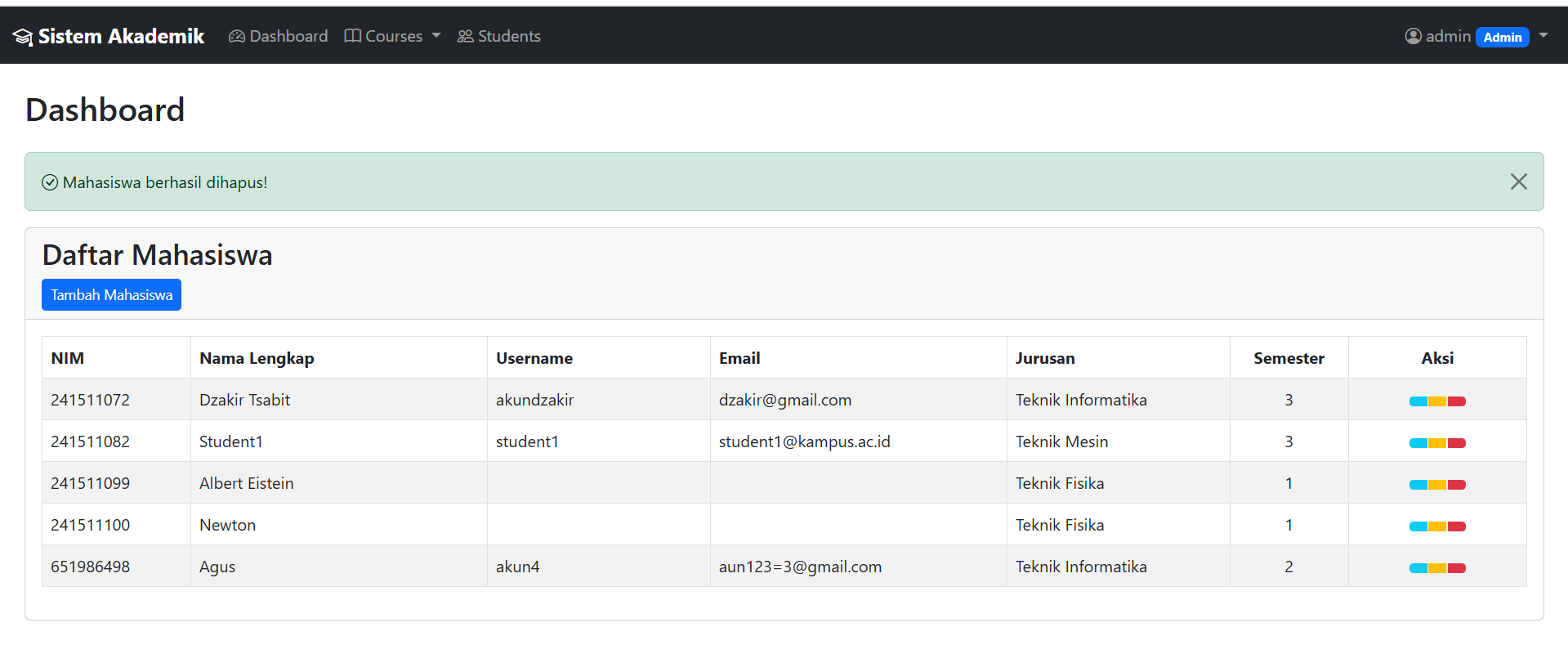
Di tombol aksi ada warna , jika warna biru untuk lihat lebih detail, namun detail nya belum ada, lanjut ke warna kuning, di warna kuning itu adalah untuk edit data mahasiswa:





Dan warna merah untuk menghapus data mahasiswa dengan valdation terlebih dahulu :





**Common Use Cases**

Menguasai **DOM** dan **event** memang penting. Namun, yang membedakan aplikasi biasa dengan aplikasi profesional adalah bagaimana cara menggunakannya secara cerdas. Beberapa trik umum yang sering dipakai antara lain:

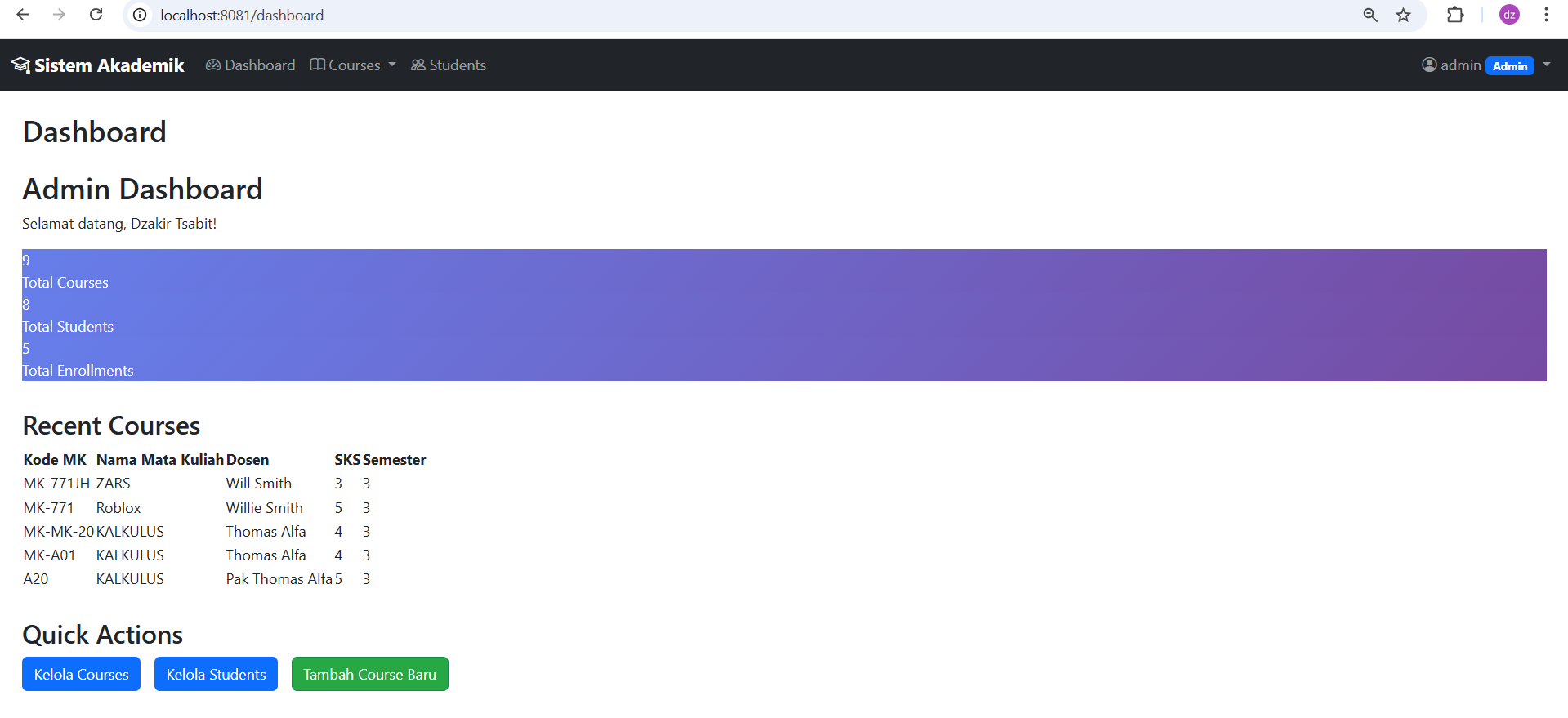
1. **Menu Aktif**  
   Menunjukkan halaman yang sedang aktif dengan menandai menu tertentu. Hal ini membantu pengguna mengetahui posisi mereka di dalam aplikasi. Biasanya dilakukan dengan menambahkan kelas .active secara otomatis berdasarkan URL.
2. **Form Validation**  
   Memberi umpan balik langsung jika ada data yang kurang atau salah saat pengguna mengisi formulir. Misalnya, dengan menandai input yang salah menggunakan garis merah. Cara ini meningkatkan pengalaman pengguna dan sekaligus mengurangi beban server.
3. **Delete Confirmation**  
   Menampilkan dialog konfirmasi sebelum menghapus data penting, agar pengguna benar-benar yakin dengan tindakannya. Contoh penerapannya adalah dengan onclick="return confirm()", yang dapat mencegah kesalahan penghapusan.

**Kaitan dengan Definition of Done (DoD):**

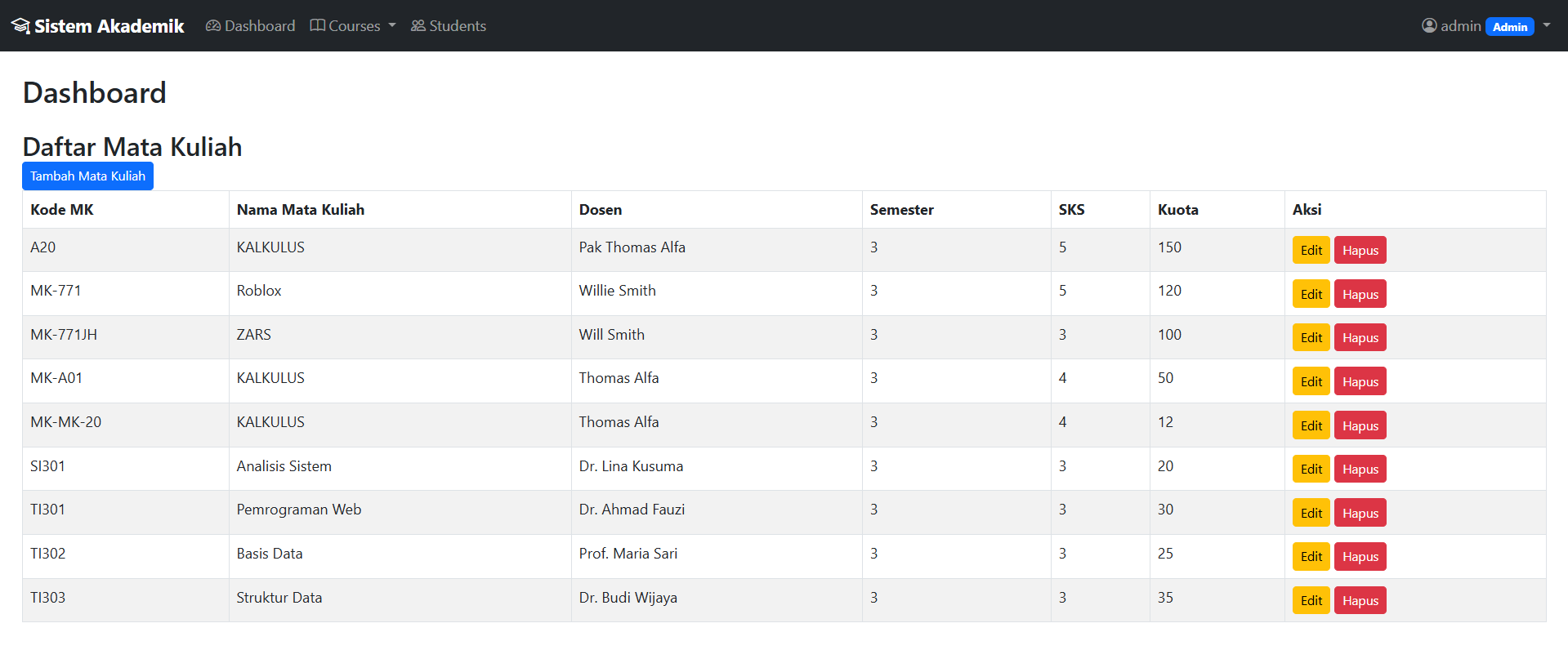
Implementasi ini adalah inti dari pemenuhan DoD terkait kualitas Antarmuka Pengguna (UI):

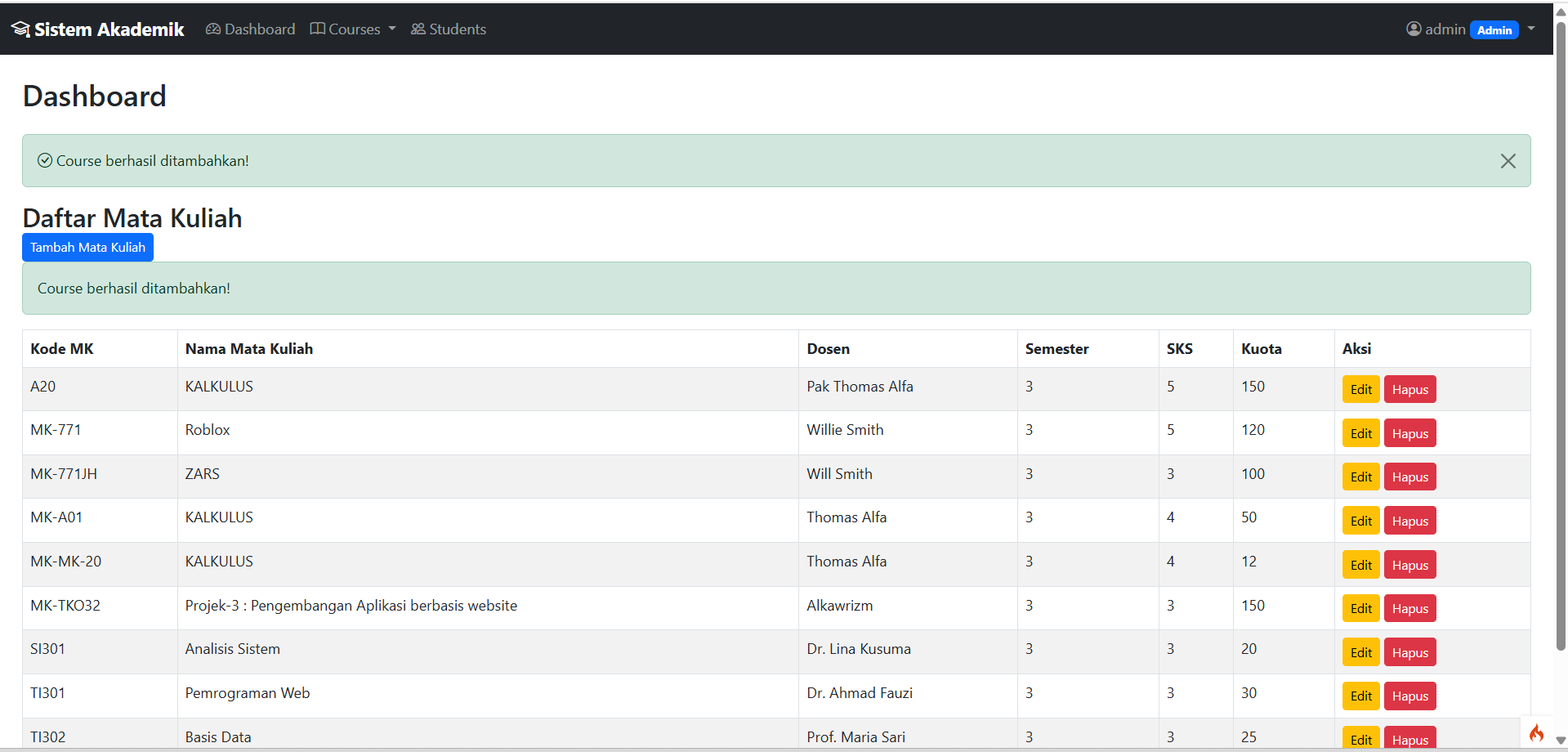
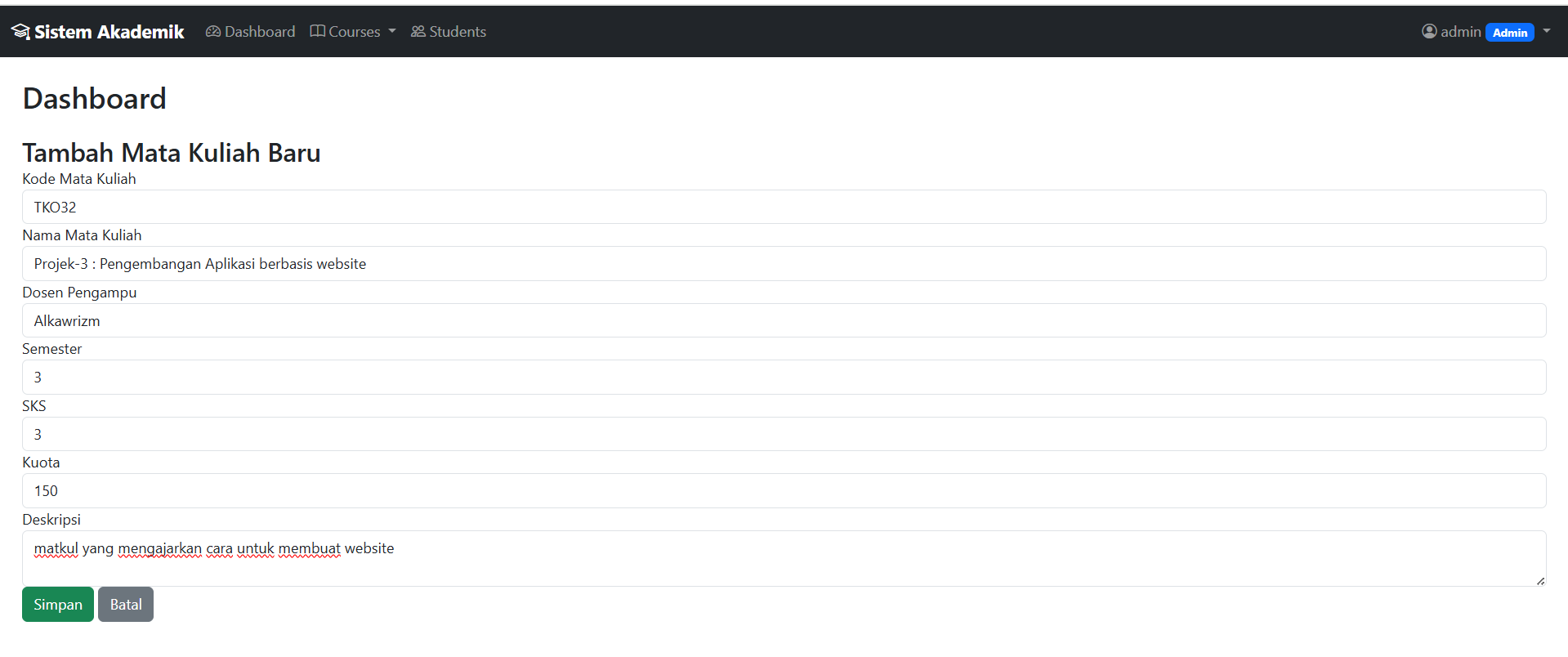
* UI memiliki: Menu aktif saat dipilih.
* UI memiliki: Validasi form dengan error message & border merah.
* UI memiliki: Konfirmasi sebelum delete

**UI\_Active\_Menu**

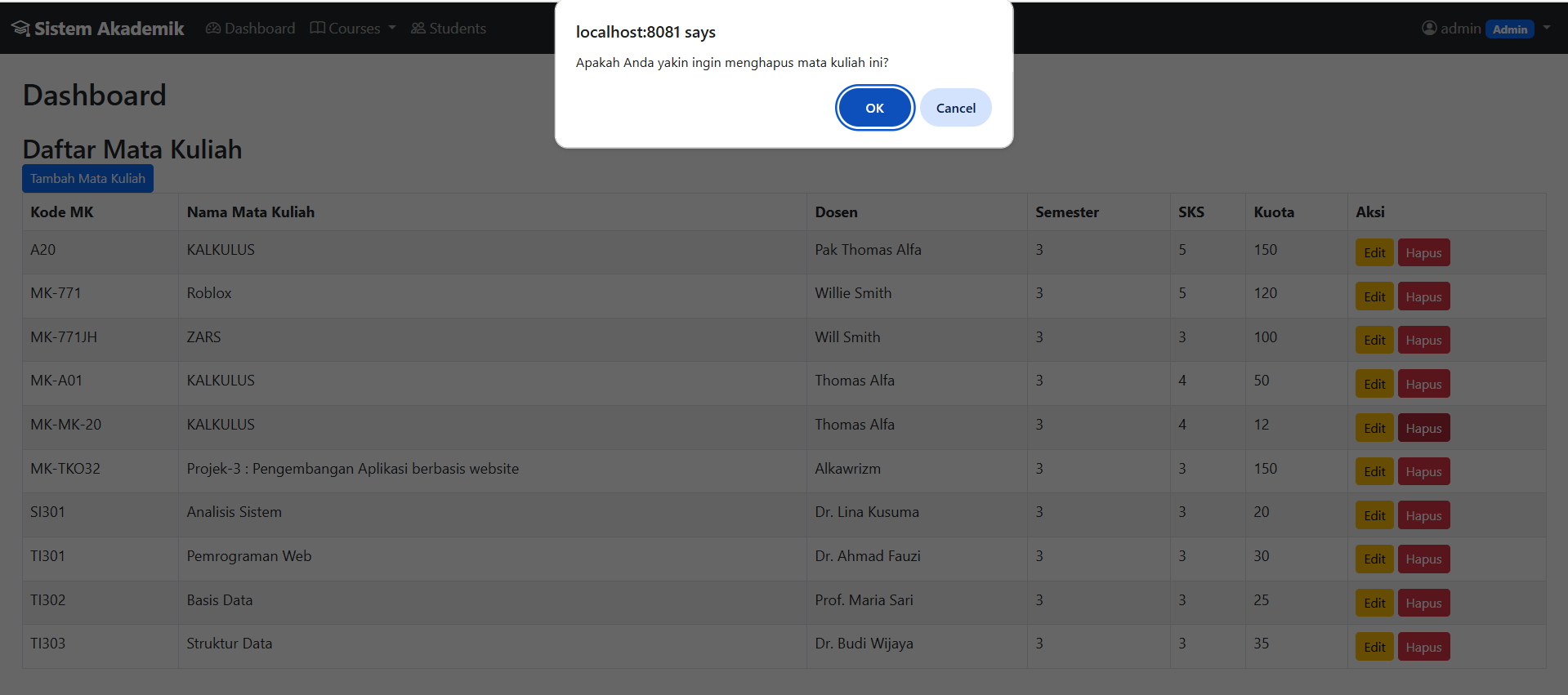


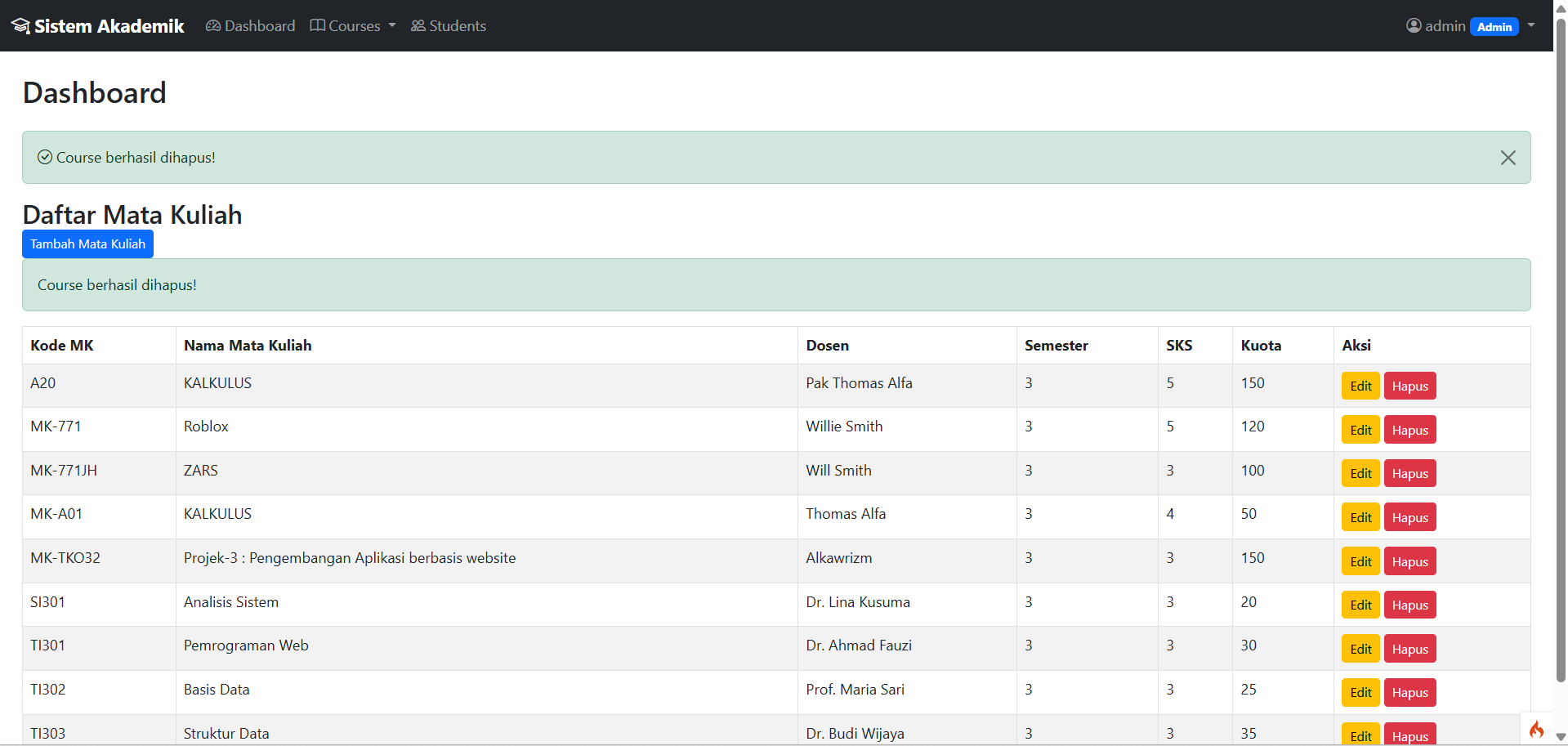
**UI\_Form\_Validation :**

****

****

**UI\_Delete\_Confirmation**

****

****

**Sync vs Async**

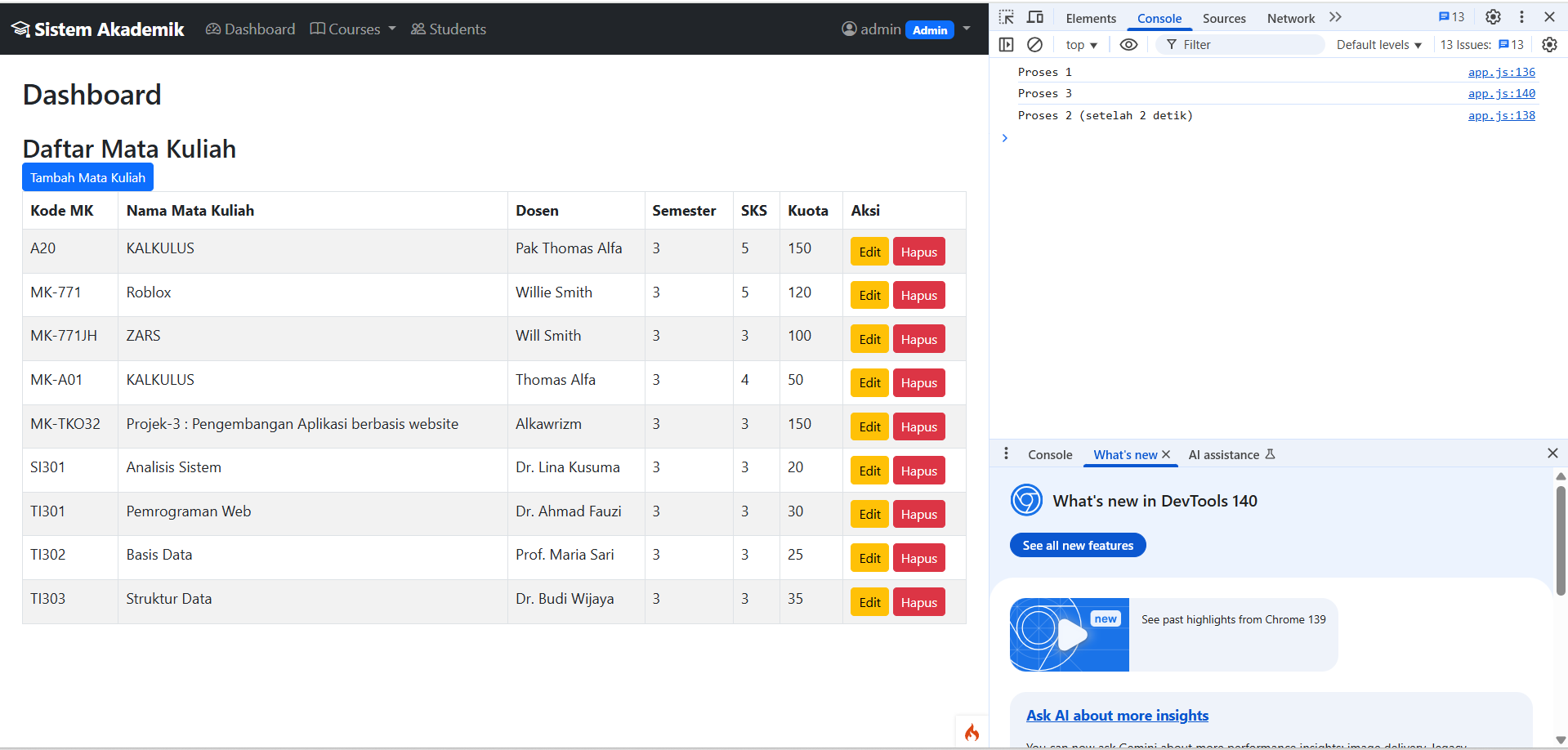
Berbeda dari cara tradisional, di aplikasi web modern, kita bisa melakukan banyak hal sekaligus tanpa membuat halaman berhenti bekerja. Kalau dulu saat kirim data (submit), browser menunggu server membalas dan halaman jadi tidak responsif, sekarang kita bisa mengirim data dan tetap bisa berinteraksi dengan halaman lainnya. Contohnya, pakai setTimeout, kita bisa menunda menjalankan fungsi tertentu selama beberapa detik, sambil pengguna tetap bisa melakukan hal lain. Ini langkah awal memahami konsep operasi asinkron, yang nantinya digunakan untuk berkomunikasi dengan server di latar belakang tanpa mengganggu pengalaman pengguna

**aitan dengan Definition of Done (DoD):**

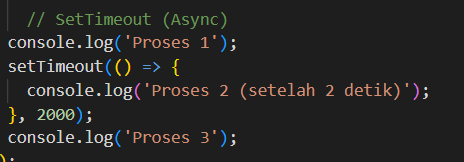
Ini adalah pemenuhan syarat materi, bukan fungsional, namun sangat penting:

* Perlihatkan contoh setTimeout (async).
* Siapkan untuk materi REST API di pertemuan berikutnya.

**Async\_setTimeout\_Proof**

****

Didalam kode saya hanya seperti ini, public/app.js:



**Lesson learn :**

**1. Array of Objects: Bukan Sekadar Teori, tapi Cetak Biru Data**

Sebelum *ngoding* tampilan, biasakan memikirkan bentuk data terlebih dahulu. Menyiapkan data mahasiswa atau mata kuliah dalam format array of objects di PHP (controller) adalah langkah fundamental. Ini seperti menyiapkan cetak biru (blueprint) yang rapi, sehingga saat ingin menampilkannya di HTML (view) menggunakan foreach, prosesnya menjadi sangat logis dan terstruktur.

**2. DOM Selector & Manipulation: Kunci Membuat UI "Hidup"**

JavaScript memberi saya kemampuan untuk mengubah elemen HTML secara langsung tanpa *reload*. Daripada hanya menampilkan halaman statis, saya bisa secara aktif "menyentuh" dan memanipulasi elemen. Contoh paling jelas adalah kalkulator SKS di halaman *enrollment*. Setiap *checklist* dicentang, innerText dari total SKS langsung di-update. Inilah inti dari User Experience (UX) yang modern; aplikasi terasa responsif dan instan.

**3. Event Handling: Mengubah Halaman dari Pasif Menjadi Responsif**

addEventListener adalah cara membuat aplikasi "mendengarkan" aksi pengguna.saya tidak lagi menunggu pengguna menekan tombol *submit* untuk melakukan sesuatu. Dengan "mendengarkan" *event* seperti change pada *checkbox* atau click pada tombol,saya bisa memicu fungsi JavaScript secara *real-time*. Ini mengubah interaksi dari yang tadinya kaku (tunggu-reload-lihat hasil) menjadi sebuah dialog yang cair antara pengguna dan sistem.

**4. Common Use Cases: Belajar dari "Bug" dan Detail Kecil**

Fitur seperti **menu aktif**, **validasi form**, dan **konfirmasi hapus** adalah detail yang membedakan aplikasi biasa dengan aplikasi yang profesional. Namun, di sinilah saya sering menemukan keanehan /bug kecil. Misalnya, setelah berhasil menghapus data, terkadang muncul dua notifikasi atau notifikasi tidak hilang. Ini adalah pelajaran berharga tentang sinkronisasi antara *backend* dan *frontend*. Kemungkinan besar ini terjadi karena cara *session flash data* dari CodeIgniter di-render dan bagaimana halaman di-*redirect*. Menemukan dan memperbaiki *bug* kecil seperti ini justru mengajarkan kita untuk lebih teliti terhadap alur data dan *state management* aplikasi

**5. Sync vs Async: Memahami Konsep "Menunggu" vs "Sambil Jalan"**

setTimeout memberikan saya gambaran dasar tentang operasi **asinkron**. Saat ini, sebagian besar aksi saya masih **sinkron** (tekan *submit*, halaman *reload* dan "menunggu" respons server). Konsep *async* mengajarkan saya bahwa ada cara untuk melakukan sesuatu "sambil jalan" misalnya, meminta data ke server di latar belakang tanpa membuat seluruh halaman macet. Meskipun di misi ini hanya sebatas contoh di konsol, ini adalah fondasi penting untuk memahami cara kerja REST API dan membangun aplikasi *single-page* yang super cepat kedepannya