

Model View Controller (2nd Edition)

Gabriel Voicu

Agenda

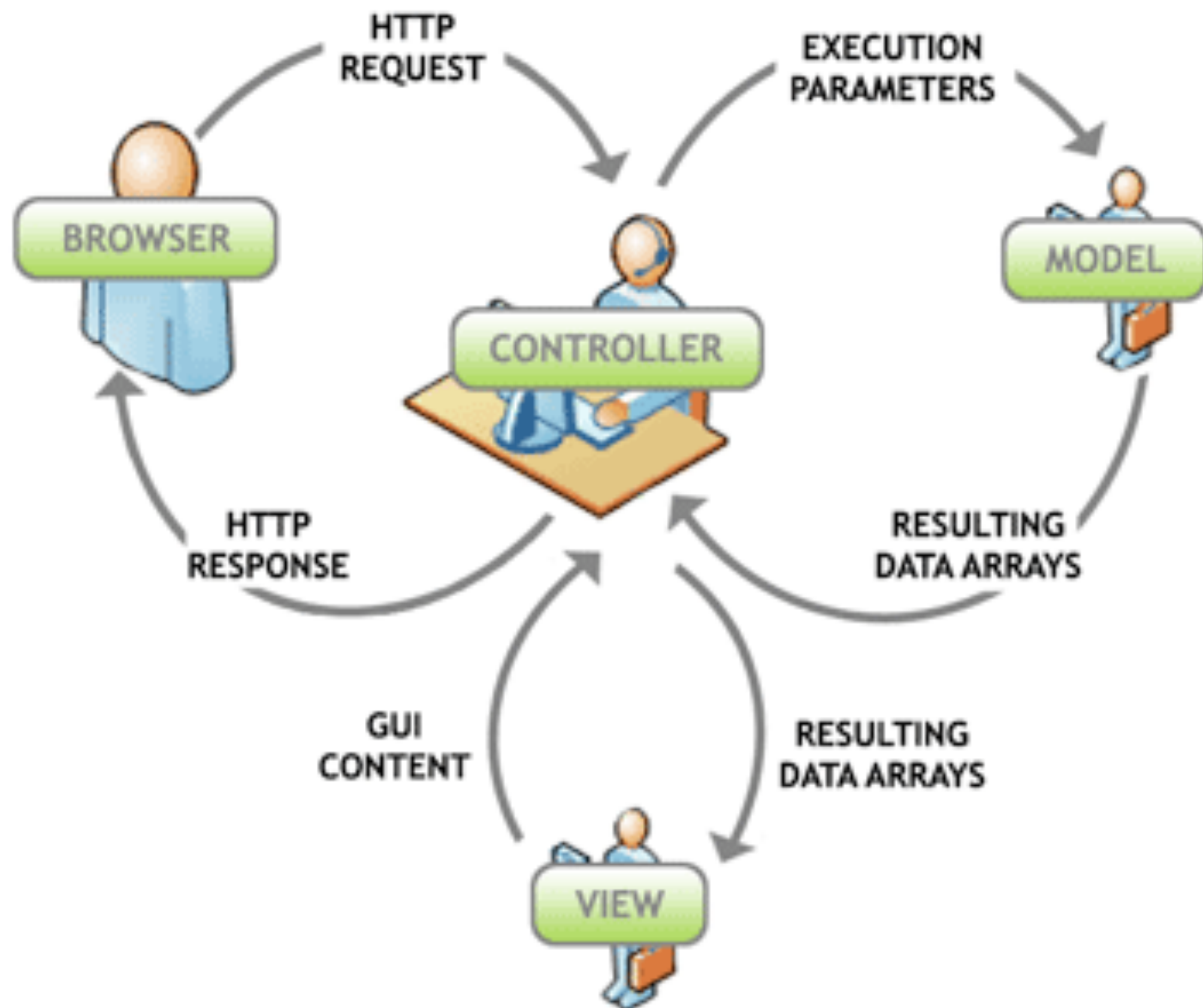
- Remember ModelViewController
- SQL
- ModelViewController
- Cookies / Sessions
- Autenticare utilizatori

ModelViewController

- De ce?
 - Structurarea codului
 - Flexibilitate
 - Ușor de iterat
 - Mai puține bătăi de cap

ModelViewController

- Controller
 - Locul unde se găsește logica aplicației
 - Realizează comunicarea între Model și View
- View
 - Locul unde se comunică cu utilizatorul
 - În Ruby, comunicarea Controller - View se realizează printr-un sistem de template-uri



ModelViewController

- Dar înainte de asta... puțin despre baze de date





SQL

- SQL (Structured Query Language)
- Datele sunt organizate sub formă de tabele, care simbolizează o entitate
- Fiecare coloană a tabelului are un tip și reprezintă un atribut ce-l definește pe acesta
- Fiecare element din tabel reprezintă

Create tabel

- CREATE TABLE Students
(
id int NOT NULL PRIMARY KEY,
last_name VARCHAR(255),
first_name VARCHAR(255),
nota_web_dev int
)

Inserare / Editare

- `INSERT INTO Students VALUES (0, 'Gabriel', 'Voicu', 4);`
- `UPDATE Students SET nota_web_dev = 5;`

Filtrare de date

- `SELECT first_name FROM Students;`
- `SELECT * FROM Students;`
- `SELECT * FROM Students WHERE
nota_web_dev=5;`

Member of Congress	
member name	state name
	<i>FK</i>
Tom Lantos	California
Tom Cambell	California
Bella Absug	California



State		
state name	state code	row id
<i>CK</i>	<i>CK</i>	<i>CK</i>
Alabama	AL	1
California	CA	2
New York	NY	3
Tennessee	TN	4

- ALTER TABLE Students
ADD varsta int;
- ALTER TABLE Students
DROP varsta;
- DELETE FROM Students
WHERE nota_web_dev < 5 and
varsta = 20;

Joins

- <http://www.codinghorror.com/blog/2007/10/a-visual-explanation-of-sql-joins.html>

Exerciții

- Conectați-vă la baza de date
- Creați tabelul Student
- Inserați-vă pe voi și pe colegul vostru în tabel;
- Afișați numele vostru mic;



Model

- SINGURUL loc unde codul tău trebuie să comunice cu baza de date
- Modelul va comunica cu baza de date printr-un ORM (Object Relational Mapping) sau prin query-uri directe in baza de date

ORM

- Fiecare bază de date, chiar dacă e de tip *SQL, are diferențe de limbaj
- Acest lucru poate crea probleme dacă se dorește migrarea proiectului la o altă bază de date
- ORM-ul creează un strat de persistență între baza de date și Model

ORM

- Cum?
 - Fiecare tabel este reprezentat ca o clasă
 - Reprezintă o clasă asociată unui tabel. Coloanele din tabel vor fi accesibile ca attribute ale clasei
 - Fiecare instanță a clasei va reprezenta un rând din tabel

ActiveRecord

- Cel mai folosit ORM în Ruby world
 - Vine inclus în Ruby on Rails
- Controller-ul va accesa baza de date apelând metode ale controller-ului

SQL vs ActiveRecord

- `Student.create!(:first_name => "Gabriel", :last_name => "Voicu", :nota_web_dev => 5)`
- `Student.where(:nota_web_dev => 5)`
- `repetent = Student.where(:nota_web_dev => 3)[0]`
`repetent.update_attributes(:nota => 3)`
`SAU`
`repetent.nota = 3`
`repetent.save`

- Student(first_name string, last_name string)
- class Student < ActiveRecord::Base
 belongs_to :series
 has_many :grades
 before_save :create_acronym
 validates_presence_of :first_name, :last_name

 def create_acronym
 self.acronym = self.first_name[0] + self.last_name[0]
 end

 def self.calc_average_grades(year)
 # self.notes = Note.where(:student_id => self.id)
 note = self.notes.where(:year => year)
 return 0 if note.size == 0
 return note.sum(:note) / note.size
 end
end

- `class Grade < ActiveRecord::Base`
 `belongs_to :student`
 `end`
- `class Serie < ActiveRecord::Base`
 `has_many :students`
 `end`

Migrații

- Migrațiile reprezintă un mod organizat prin care se poate modifica structura bazei de date
- Fiecare comandă ce modifică baza de date este scrisă într-un fișier. Prin executarea unei comenzi în terminal, fiecare fișier va fi citit pe rând și se va modifica baza de date

Migrații

- Avantaje:
 - Dacă vreți să replicați schema bazei de date pe un alt server, copiați fișierele cu migrațiile, rulați o comandă și gata. :-)
 - Învățați o singură dată cum se face

Migrații

- Exemple pentru ActiveRecord:

- Creare tabel:

```
create_table :students do |t|  
  t.string :first_name  
  t.string :last_name  
end
```

- Adaugare coloană în tabel:

```
add_column :students, :grade, :integer
```


Migrații

- Rulare:
 - rake db:migrate

Exerciții

- Decomentați migrațiile din fișierul “db/migrate” și rulați comanda “rake db:migrate”
- Creați o migrație care creează modelul “Response”, care să aibă attributele: content (text), rating (integer), created_at (datetime), user_id (integer)

Exerciții (2)

- Adăugați relația `has_many <-> belongs_to` între `Question` și `Response`
- Modificați codul din miniflow, astfel încât răspunsurile să se salveze în baza de date

Take a Break Buddy!



©123Greetings.com

Cookie-uri vs Sesiune

- Ambele sunt hash-uri care sunt folosite pentru a stoca date
- Cookie-urile stochează datele pe client în browser (pot fi blocate de utilizator)
- sesiunile sunt păstrate pe server, fiind asociate unui ID unic asociat fiecărui utilizator (nu pot fi blocate de utilizator)

Cookie-uri vs Sesiune

- La ce sunt folosite?
- Cookie-urile pot fi folosite pentru a ține minte anumite acțiuni pe care utilizatorul le-a realizat în primă instanță și pe care vrem să nu le mai repete
- Exemplu: s-a autentificat odată în aplicația noastră și vrem ca la accesarea viitoare a aplicației, acesta să nu-și mai introducă datele de autentificare => le păstrăm în cookie-uri

Cookie-uri Vs Sesiune

- Datele stocate în sesiune persistă cât timp utilizatorul se află pe pagină. După ce acesta a închis pagina, aceste date se pierd.
- Exemplu de folosire: dacă un utilizator trebuie să completeze un formular format din mai multe ecrane, poate reține pași alternativi în sesiune.

Autentifcare



SECURITY
YOU'RE DOING IT WRONG

Întrebări?

