Università della Svizzera italiana

Faculty of Informatics

Express.JS

Prof. Cesare Pautasso
http://www.pautasso.info
cesare.pautasso@usi.ch
@pautasso

Modularity

./my_module.js

```
var msg = "x:"; //private
var f = function(x) {
  return msg + " " + x;
}
module.exports.f = f; //public
```

Export module functions making them part of `module.exports`

```
var module = require('module'); //library module
var my_module = require('./my_module'); //local module
console.log(my_module.f("hello"));
```

Import modules with `require`

package.json

./package.json

```
{
  "name": "my-app",
  "description": "My Web Application",
  "version": "0.0.1",
  "dependencies": {
    "express": "3.x"
  }
}
```

`dependencies` lists the required modules and their version

Use the Node Package Manager (`npm`) to:

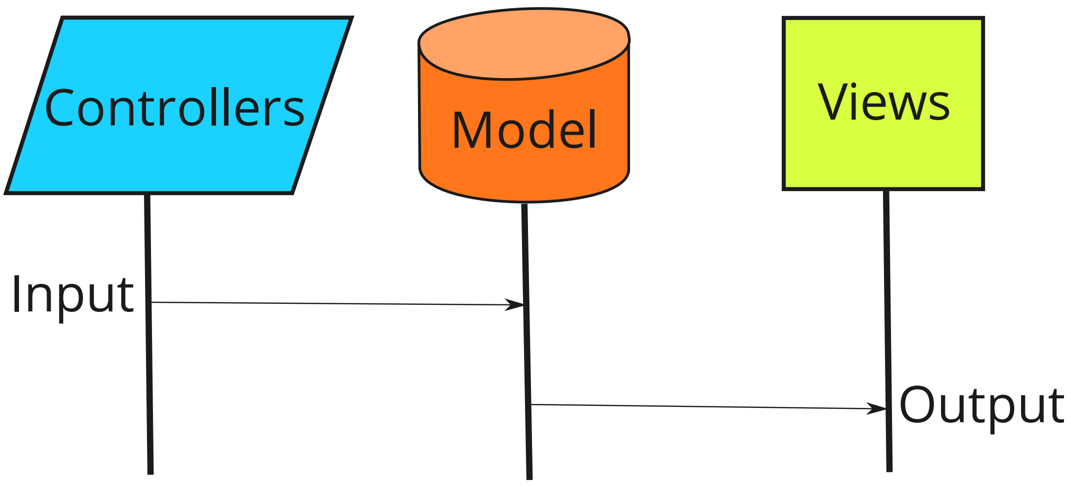
```
npm init
```

interactively create a new package.json

```
npm install
```

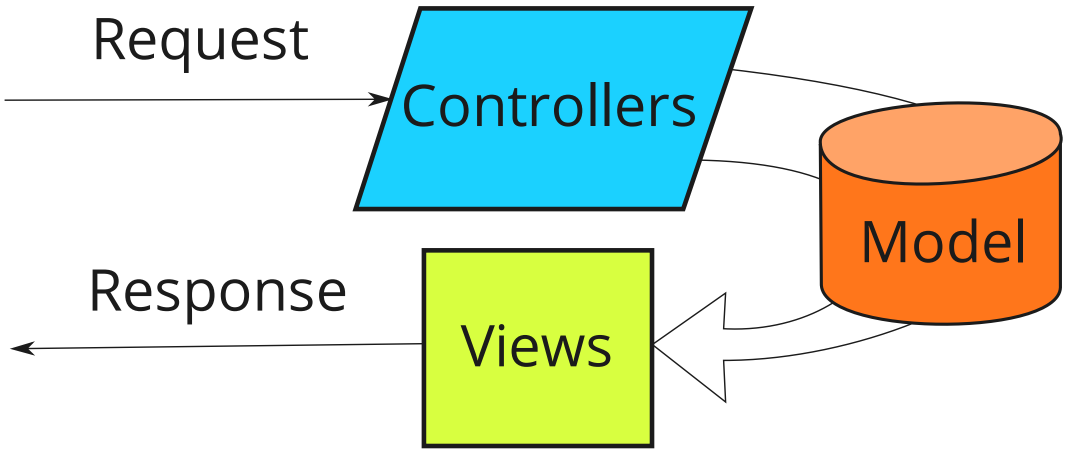
download and install the dependencies

Model View Controller



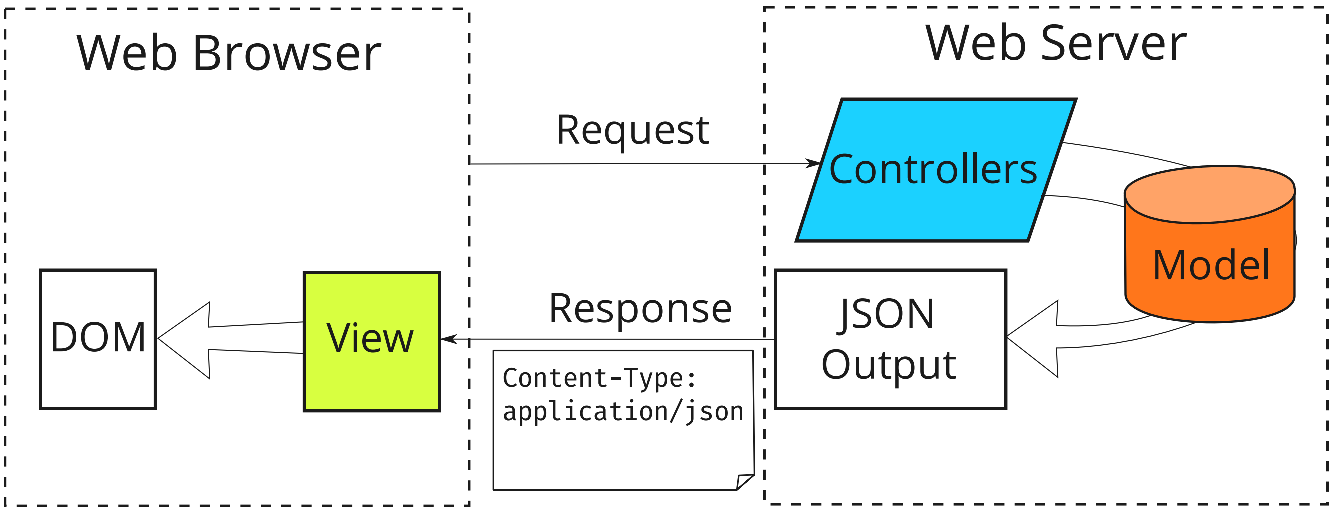
- User input is handled by the Controller which may change the state of the Model
- The Views display the current state of the Model (and may receive notifications about state changes by observing it)

MVC on the Web 1.0



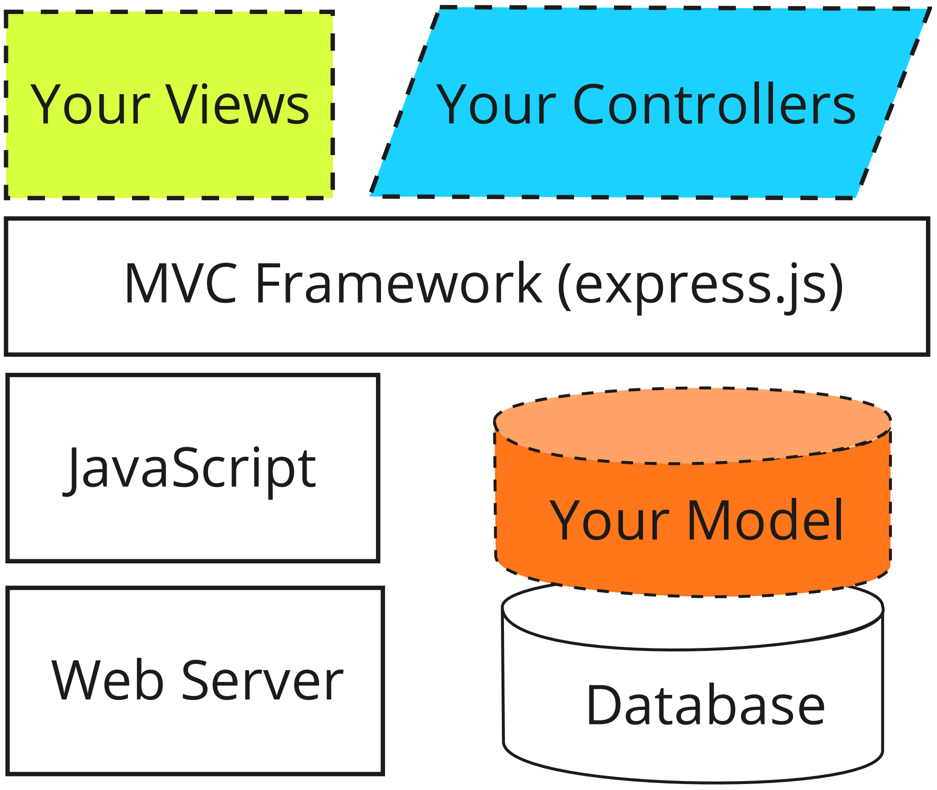
- In Web applications, views produce the actual HTML pages sent to the browser. The controller is the code that handles the input requests, reads/writes the data model and invokes the view to format the model in HTML (or other) representation. The data model manages the actual content and the state of the application, usually stored persistently in a database or XML files.

MVC on the Web 2.0



- The Web server responds with JSON data extracted from the model
- The view running on the client renders the JSON into HTML/DOM

Express MVC



Express MVC

```
var express = require('express');
var app = express();
app.get('/',
  function(req, res) {
    db.findAll(function(error, model) {
      res.render('view', model);
    })
  });
app.listen(8888);
```

Routing

```
app.METHOD('URI Template', callback)
```

METHOD = {get, put, post, delete, all}

```
app.get('/user/:id',
  function(req, res){
    res.send('user ' + req.params.id);
  });
app.get('/file/*.*',
  function(req, res){
    res.send('file path: ' + req.params);
  });
```

```
app.routes //read the configuration
```

Parse the Request Body

```
app.use(express.bodyParser());
```

```
app.put('/blog/:year/:id?', //? optional
  function(req, res){
    var year = req.params.year;
    var id = req.params.id || 0; //default 0
    var body = req.body; ...
    var files = req.files; //uploaded files
  });
```

Content-Type

```
//Request Content-Type:application/json
if (req.is("application/json")) {
  //req body is JSON
}
//Request Accept:text/html
if (req.accepts("html")) {
  //set Response Content-Type Header
  res.type("text/html");
}
```

Views

```
function(model) { return html }
```

Views are template files that present content to the user: variables, arrays and objects that are used in views are passed by the controller to be rendered by the view.

Views should not contain complex business logic; only the elementary control structures necessary to perform particular operations, such as the iteration over collections, and simple filters.

Views (Manual)

```
function(model) {
  var html = [];
  html.push('<html><body>');
  if (model.user) {
    html.push('<h2>' + model.user.name + '</h2>');
  }
  html.push('</body></html>');
  return html.join();
}
```

Views (Rendered)

```
res.render('view template', { model } );
```

Invoke a template to render the view

```
//render the index template using JADE
res.render('index.jade');
//render the index template using EJS
res.render('index.ejs');
```

```
app.set('view engine', 'ejs');
res.render('index');
```

Embedded JS Templates

app.js

```
var model = {
  user: { name: 'CP' }
};
res.render('index.ejs', model );
```

index.ejs

```
<html><body>
<% if (user) { %>
  <h2><%=user.name %></h2>
<% } %>
</body></html>
```

Result

```
<html><body>
  <h2>CP</h2>
</body></html>
```

Embedded JS Templates

```
<h1><%= title %></h1>                                   .ejs
<ul>
<% for(var i=0; i<list.length; i++) {%>
  <li><%= link_to(list[i],
                  'users/'+list[i]) %></li>
<% } %>
</ul>
```

- JavaScript between `<% %>` is executed
- JavaScript between `<%= %>` adds the result to the HTML

View Helpers

Reduce the amount of code by using these helper functions (mini-templates or widgets)

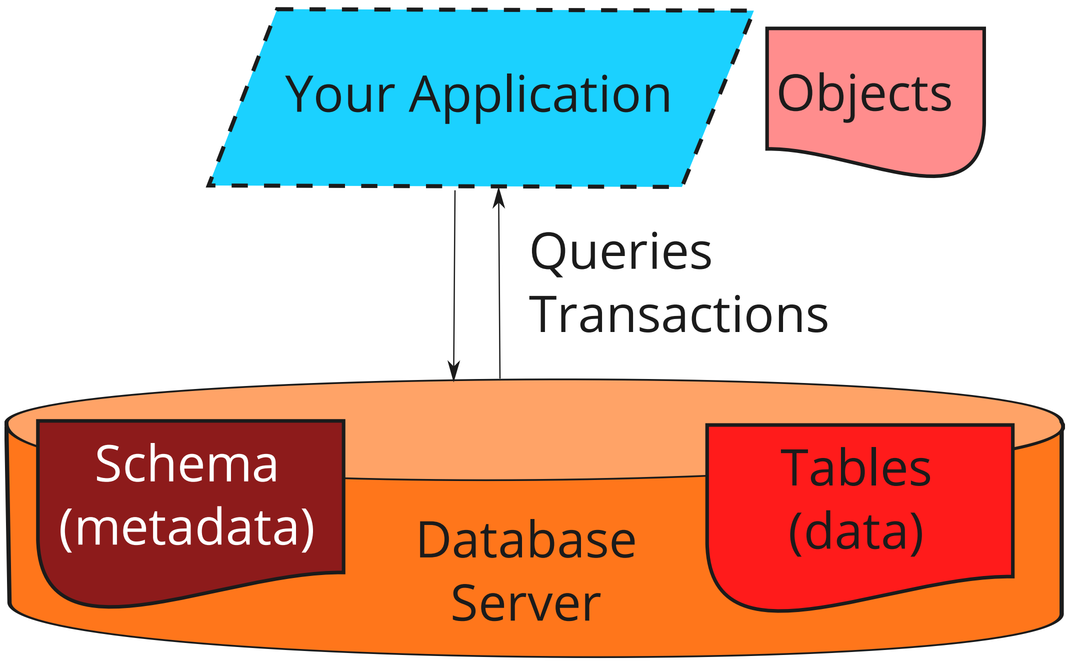
```
date_tag form_tag form_tag_end hidden_field_tag
input_field_tag is_current_page link_to
submit_link_to link_to_if link_to_unless
password_field_tag select_tag submit_tag
text_area_tag text_field_tag img_tag
```

http://code.google.com/p/embeddedjavascript/wiki/ViewHelpers

Databases

- Manage the persistent storage of structured information
- Simplify writing of programs to query the information (selection, filtering, sorting, aggregation, translation) with declarative query languages
- Guarantee consistency of the data as it is modified with concurrent transactions
- Guarantee reliability of the data thanks to advanced crash recovery features
- Very often used to implement the model of MVC Web applications on the server

Working with Databases



SQL

- Relational Databases
- Tables (Relations) store well-structured data according to a predefined Schema
- Queries defined according to the standard Structured Query Language
- Strong transactional guarantees (ACID)

NoSQL

- Non-relational Databases (Object-oriented databases, document databases, graph databases, key-value stores, XML databases)
- Schema-less (semi-structured heterogeneous data stores)
- Highly scalable with relaxed transactional guarantees and limited query expressivity

Examples

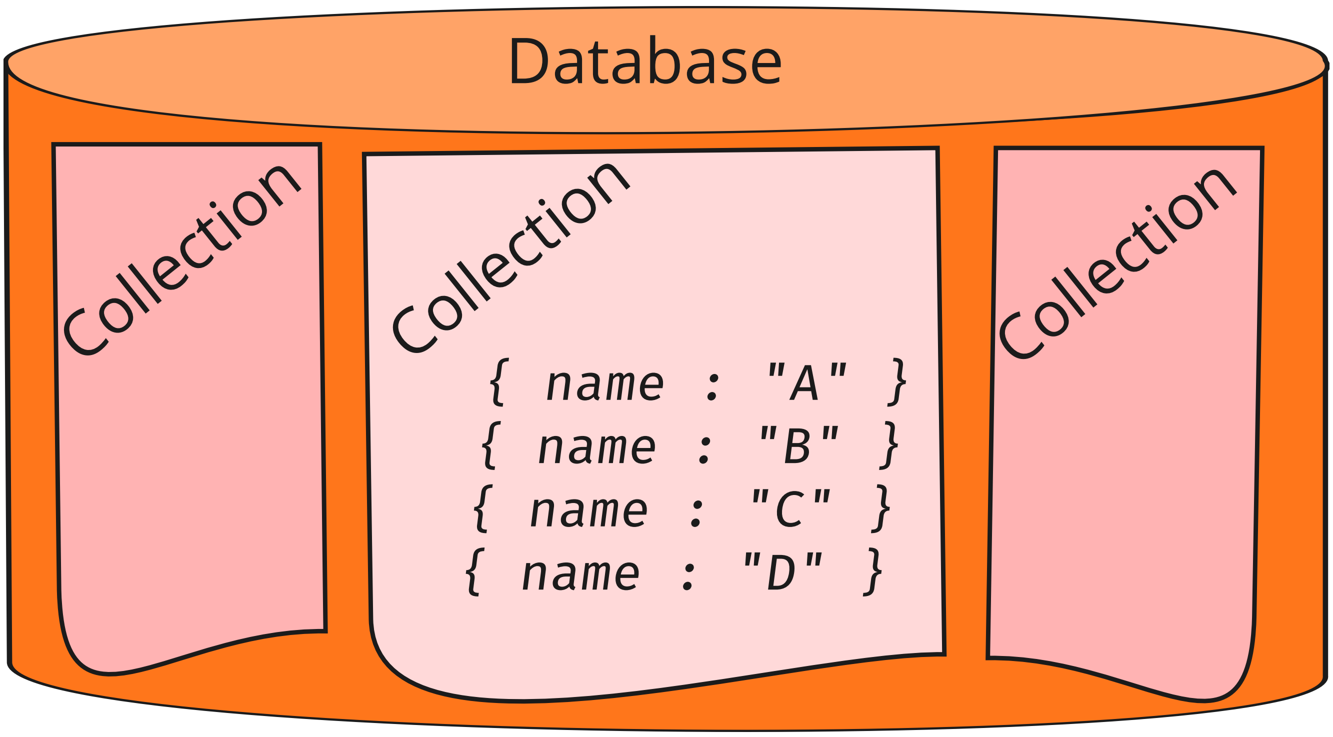
SQL

mySQL -- PostgreSQL -- Oracle -- Microsoft SQL Server -- IBM DB2

NoSQL

CouchDB -- **MongoDB** -- Apache JackRabbit -- Neo4j -- MemCacheDB -- BigTable -- Apache Hadoop

MongoDB



A collection is a set of JSON objects, which may or may not have the same structure

1. Connecting

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/db');
// The MongoDB Server should be running
// for the connection to work
```

2. Schema Definition

```
var CommentsSchema = new mongoose.Schema ({
   person     : String
 , comment    : String
 , created_at : Date
});
```

2. Schema Definition

```
var PostSchema = new mongoose.Schema({
    author : mongoose.ObjectId
  , title : String
  , body : String
  , created_at : Date
  , comments : [CommentsSchema]
});
//Create the Collection ‘Post’ using Schema
mongoose.model('Post', PostSchema);
var Post = mongoose.model('Post');
```

3. Queries

```
Post.find({},
  function (err, posts) {
        //for each post in posts
  });
Post.findById(id,
  function (err, post) {
        if (!err) {  //found post with id
    }
  });
```

4. Create and Save

```
var post = new Post(
     { title: params['title'],
        body: params['body'],
  created_at: new Date() });
post.save(function (err) { //save failed
});
```

Save also works with updates

5. Delete

```
Post.findById(id,
  function (err, post) {
    if (!err) {
      post.remove();
    }
  });
```

Tools

- Express (http://expressjs.com/) (Node.js framework)
- Node.JS (http://nodejs.org/)
- Embedded JS (http://code.google.com/p/embeddedjavascript/) (View Templating Engine)
- Mongoose (http://mongoosejs.com/) (Node-MongoDB Driver)
- MongoDB (http://www.mongodb.org/) (JSON Database)