Semantic HTML + UI

Produced by:

Eamonn de Leastar (edeleastar@wit.ie)

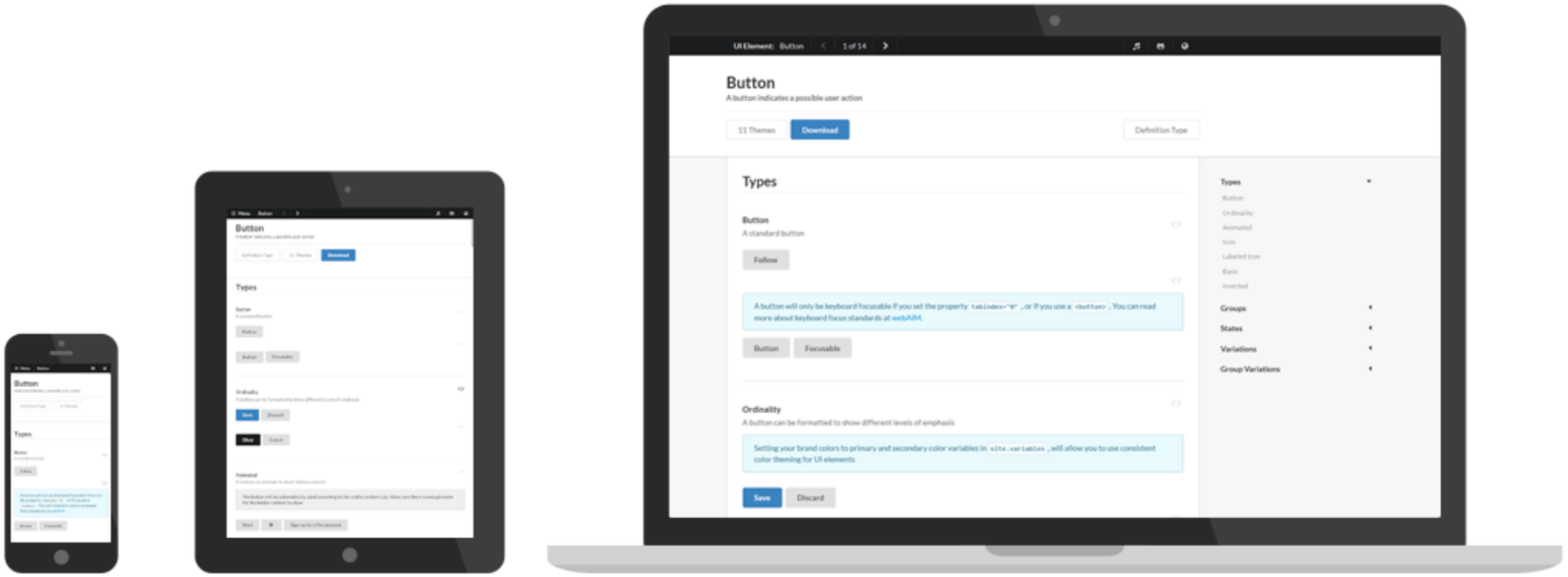
Dr. Siobhán Drohan (sdrohan@wit.ie)

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

Topics List

- Overview of Semantic
- Specifics on Semantic
- Installing Semantic with Play
- Using Semantic with Play



Button

A button indicates a possible user action

Types

Button

A standard button

A button will only be keyboard focusable if you set the property `tabindex="0"`, or if you use a `<button>`. You can read more about keyboard focus standards at webAIM.

Ordinality

A button can be formatted to show different levels of emphasis

Setting your brand colors to primary and secondary color variables in `site.variables`, will allow you to use consistent color theming for UI elements

Design Beautiful Websites Quicker

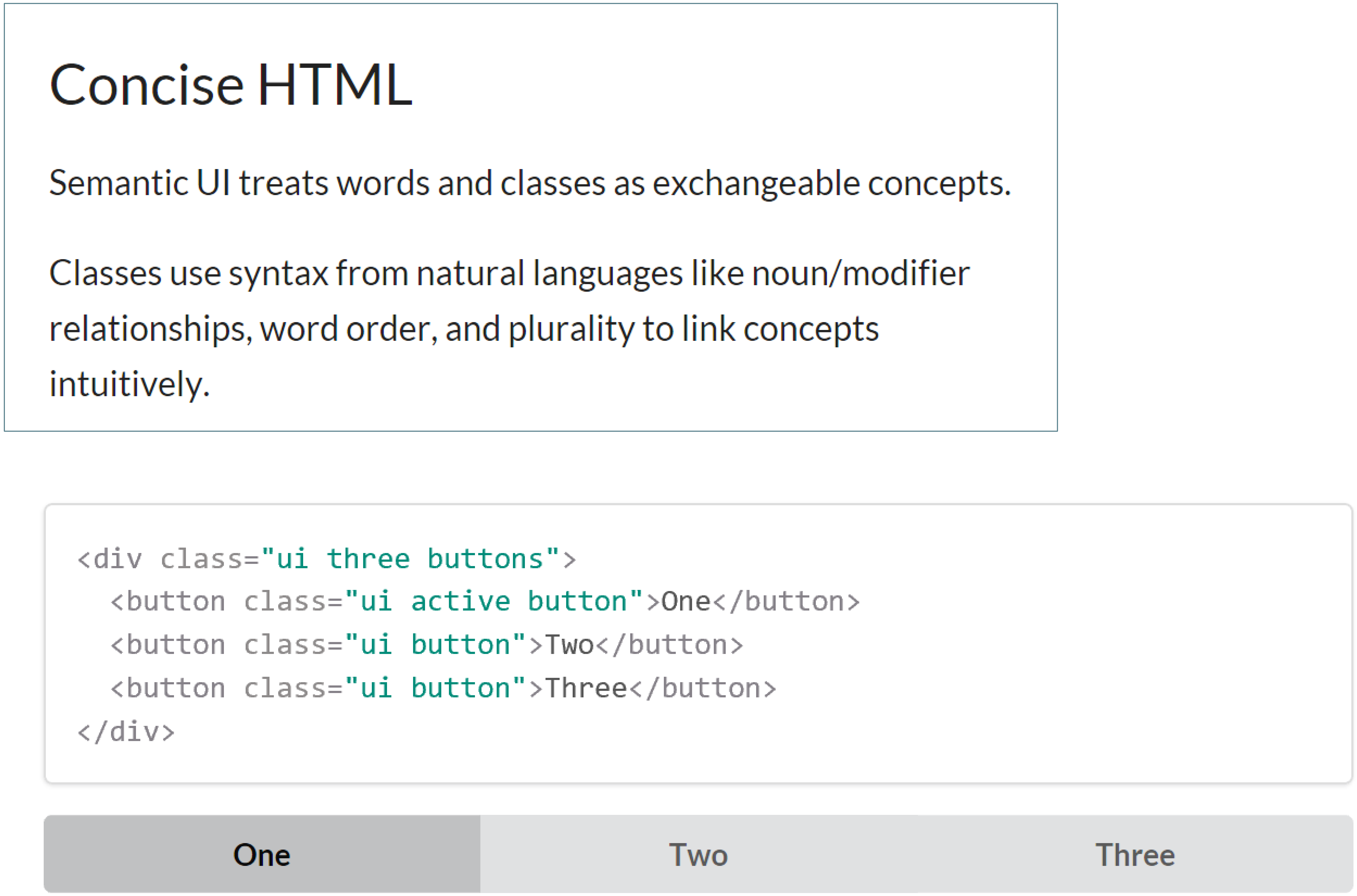
Semantic is a development framework that helps create beautiful, responsive layouts using human-friendly HTML.

http://semantic-ui.com/

Some marketing speil…

Semantic allows developers to build beautiful websites fast, with ***concise HTML****,* ***intuitive javascript****, and* ***simplified debugging****, helping make front-end development a delightful experience.*

Semantic is responsively designed allowing your website to scale on multiple devices. Semantic is production ready and partnered with frameworks such as ***React****,* ***Angular****,* ***Meteor****, and* ***Ember****, which means you can integrate it with any of these frameworks to organize your UI layer alongside your application logic.*



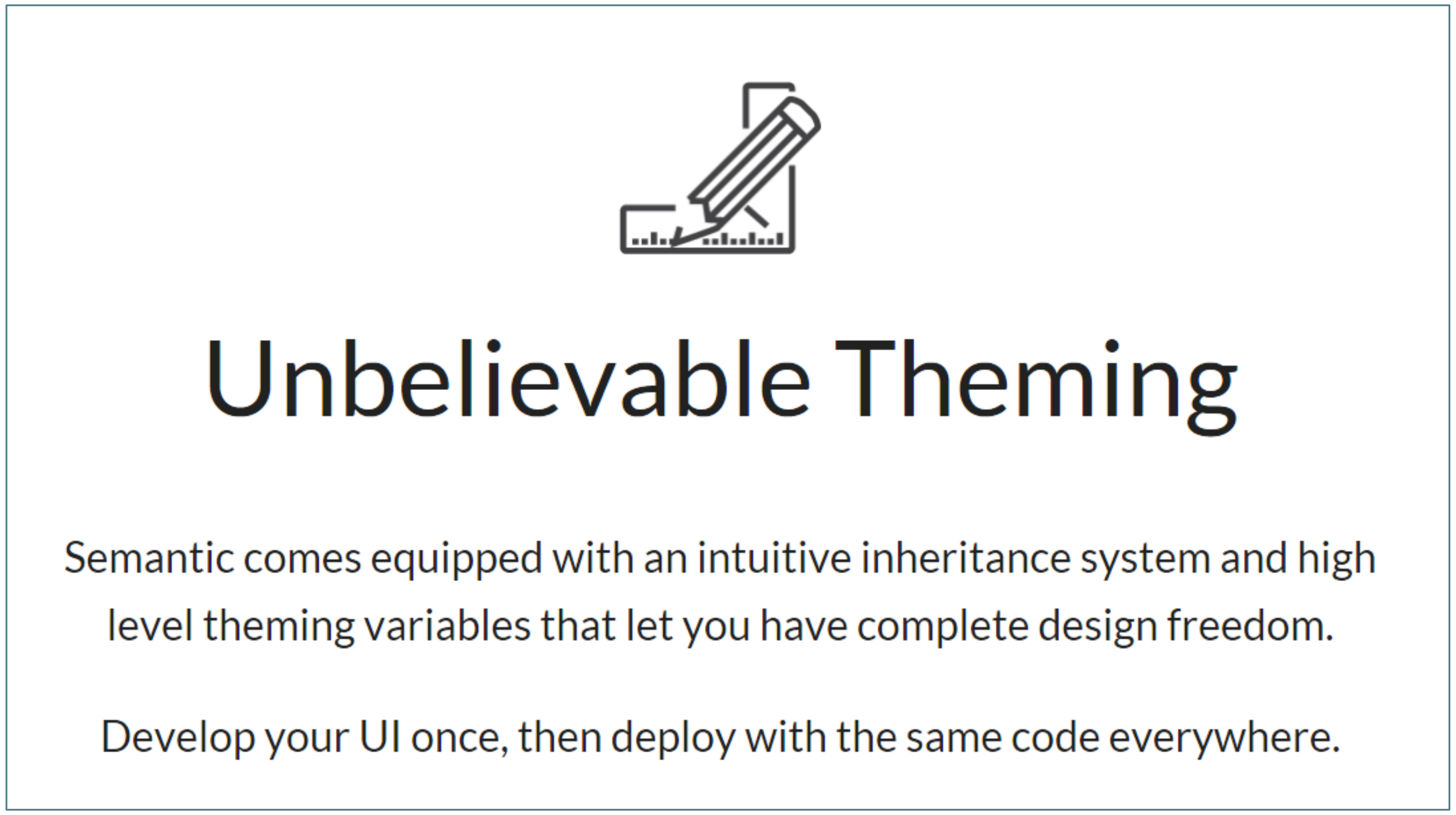
Concise HTML

Semantic UI treats words and classes as exchangeable concepts.

Classes use syntax from natural languages like noun/modifier relationships, word order, and plurality to link concepts intuitively.

```html
<div class="ui three buttons">
  <button class="ui active button">One</button>
  <button class="ui button">Two</button>
  <button class="ui button">Three</button>
</div>
```

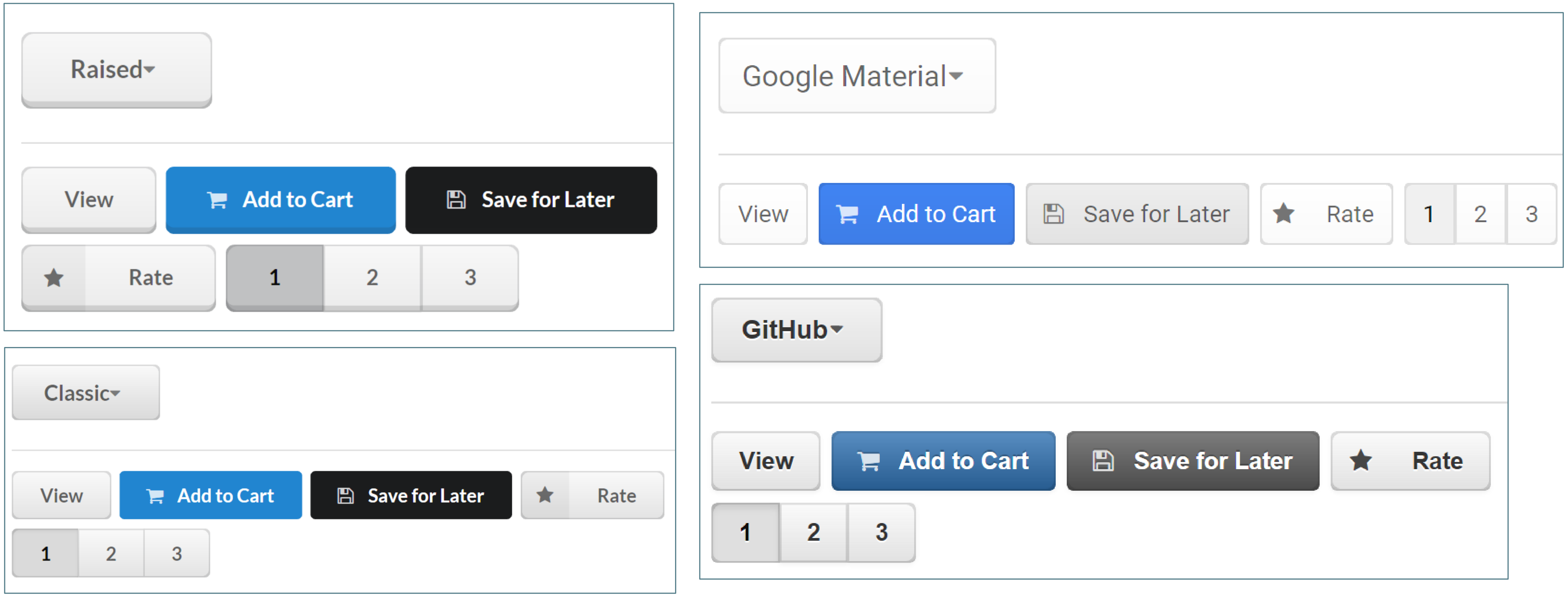
http://semantic-ui.com/



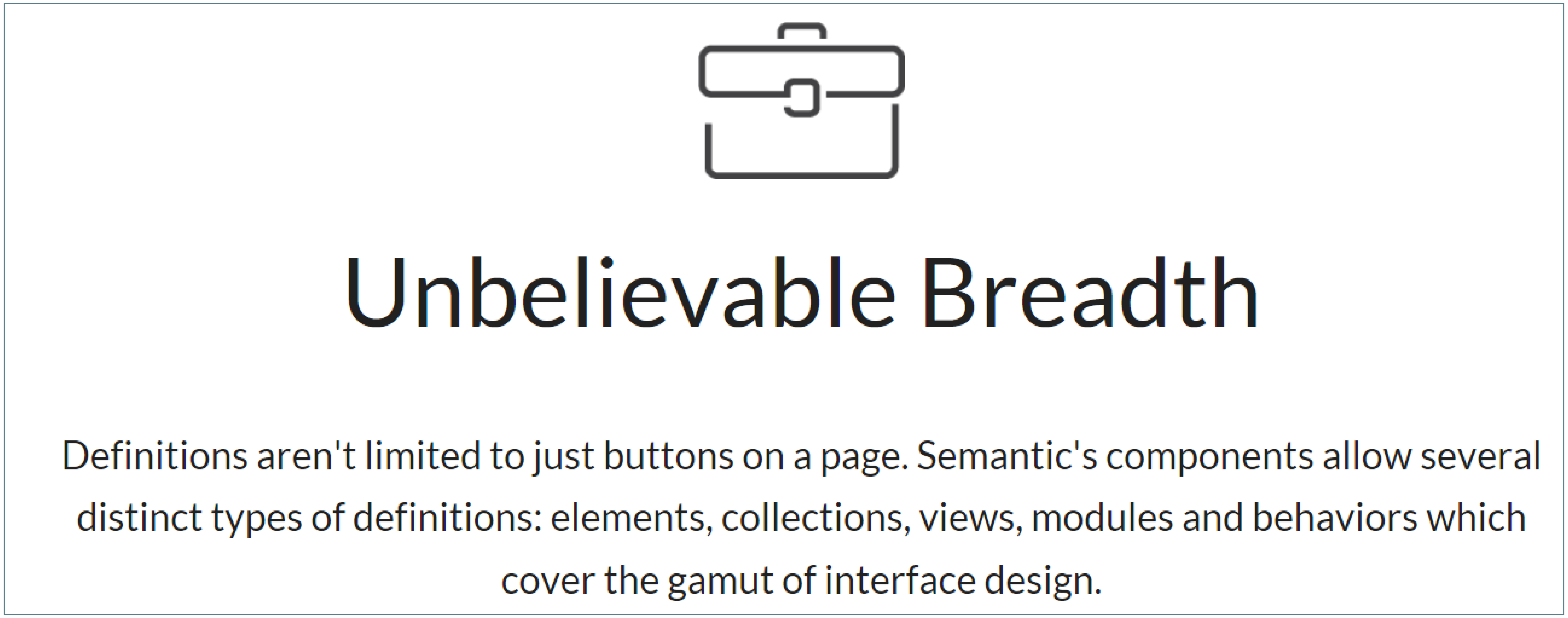
Unbelievable Theming

Semantic comes equipped with an intuitive inheritance system and high level theming variables that let you have complete design freedom.

Develop your UI once, then deploy with the same code everywhere.



Learn about changing themes: http://semantic-ui.com/usage/theming.html



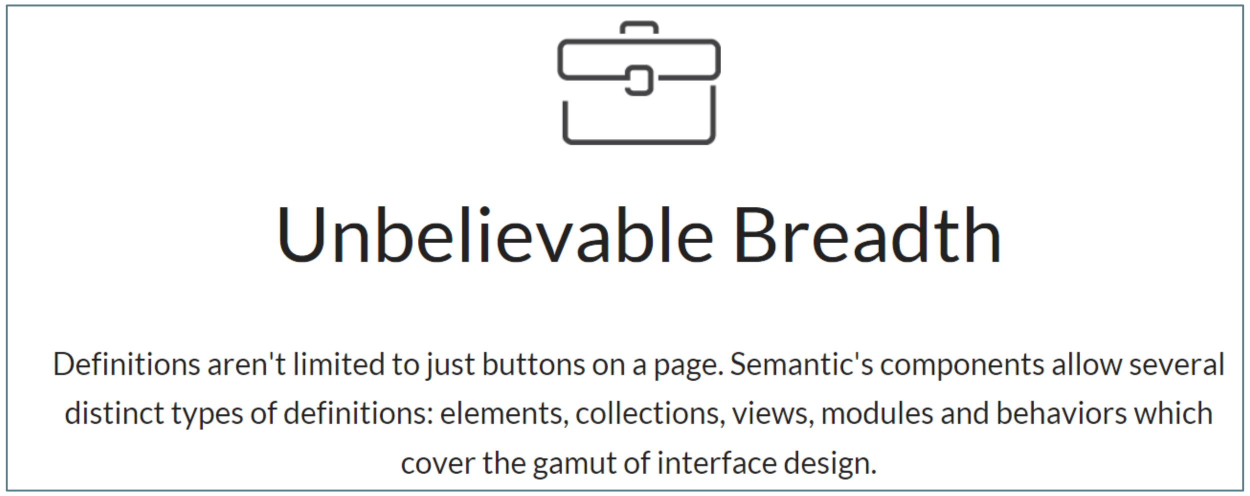
Unbelievable Breadth

Definitions aren't limited to just buttons on a page. Semantic's components allow several distinct types of definitions: elements, collections, views, modules and behaviors which cover the gamut of interface design.

http://semantic-ui.com/

Terminology – Definitions and Components

- A **definition** is a set of CSS and Javascript which describe a component's essential qualities.
- A **component** refers to any UI element packaged for distribution.
- Semantic UI classifies **components** into five areas:
  - Elements
  - Collections
  - Views
  - Modules
  - Behaviours



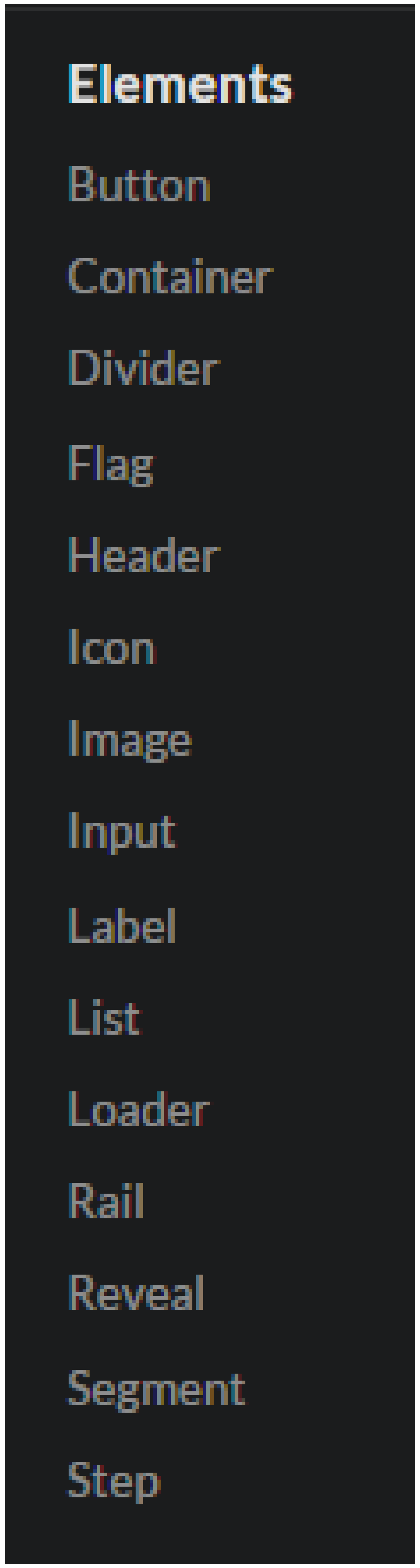
Unbelievable Breadth

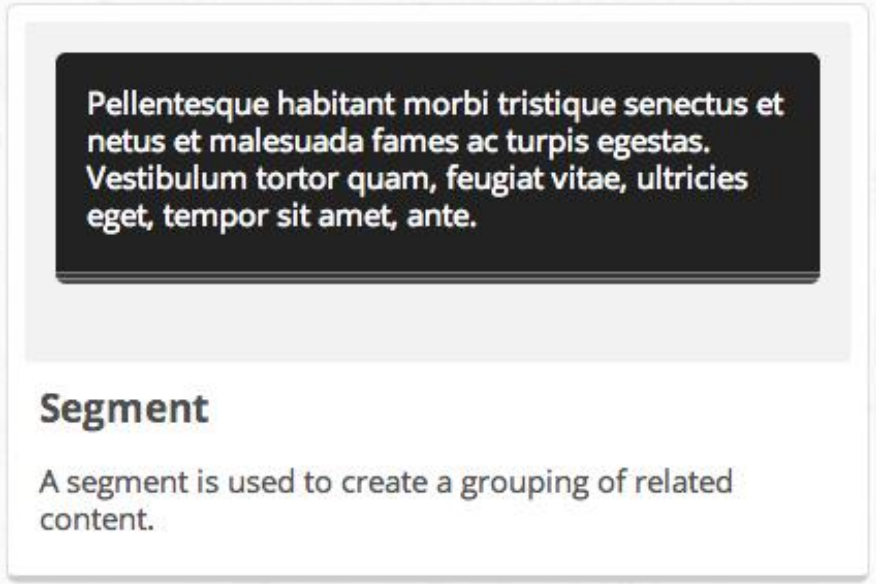
Definitions aren't limited to just buttons on a page. Semantic's components allow several distinct types of definitions: elements, collections, views, modules and behaviors which cover the gamut of interface design.

http://semantic-ui.com/introduction/glossary.html

Elements

UI elements are page elements with a single function. They can exist alone or in a plural form with elements sharing qualities e.g.: A group of buttons may use ui red buttons as a grouping with individual ui button children.

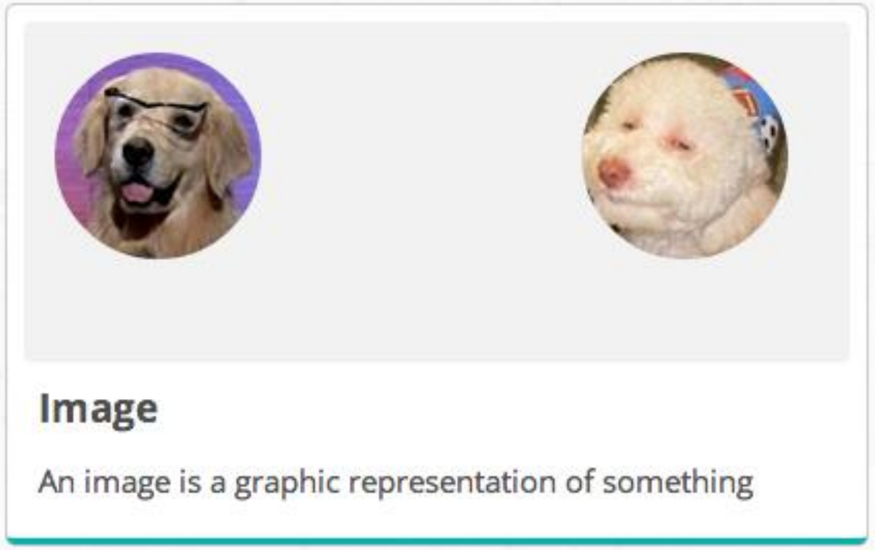




Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante.

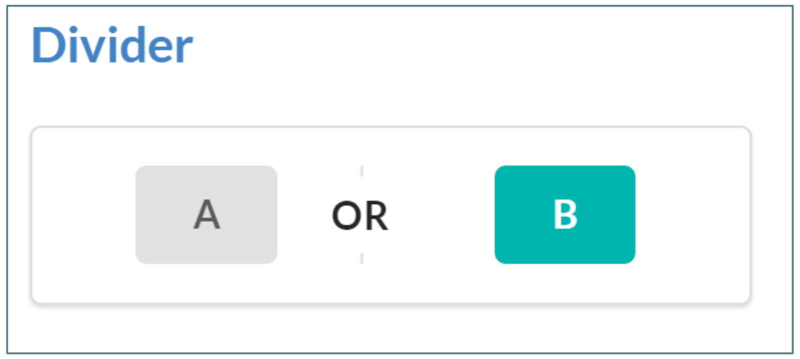
Segment

A segment is used to create a grouping of related content.

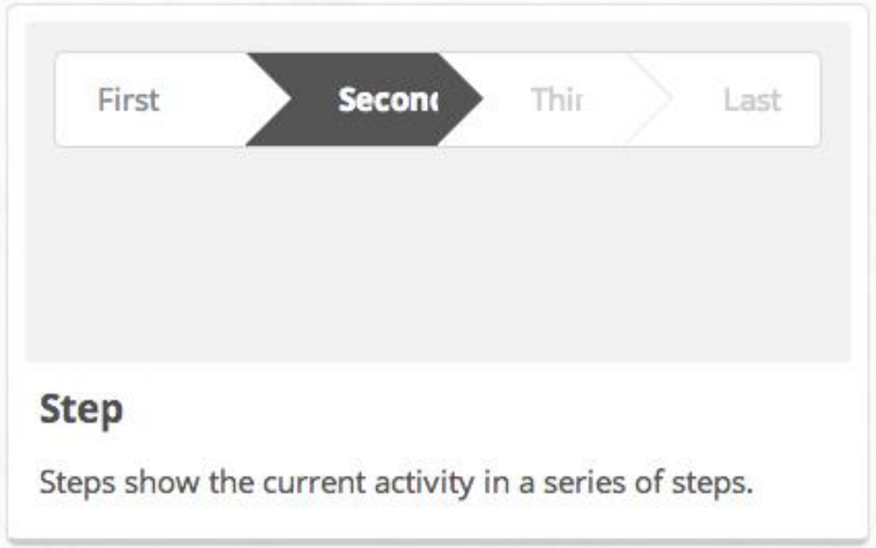


Image

An image is a graphic representation of something

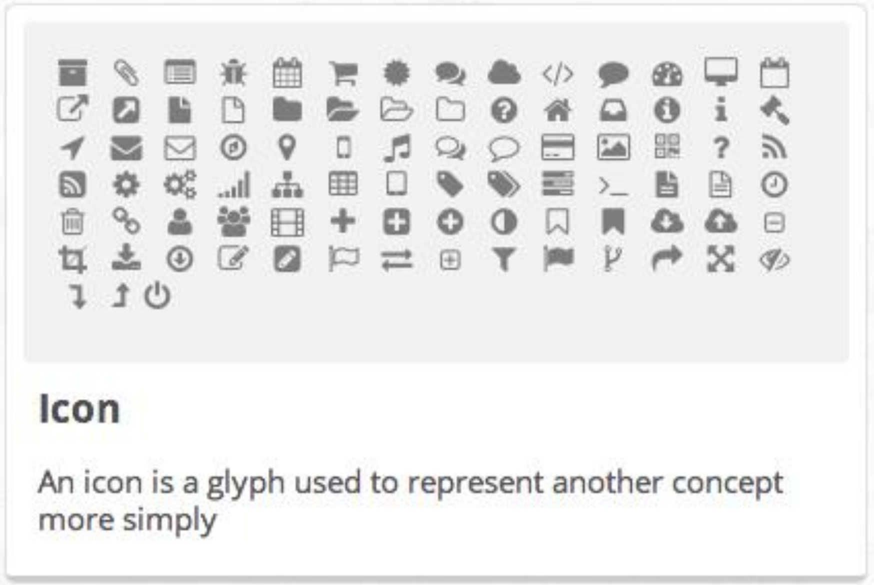


Divider



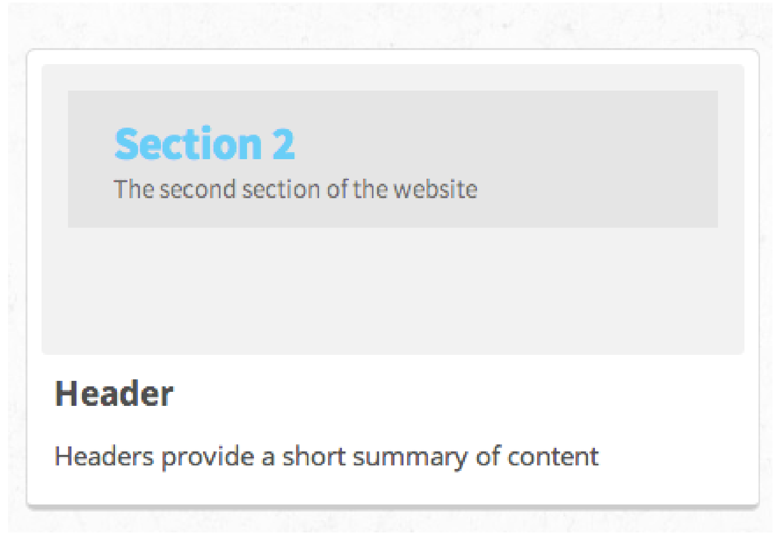
Step

Steps show the current activity in a series of steps.



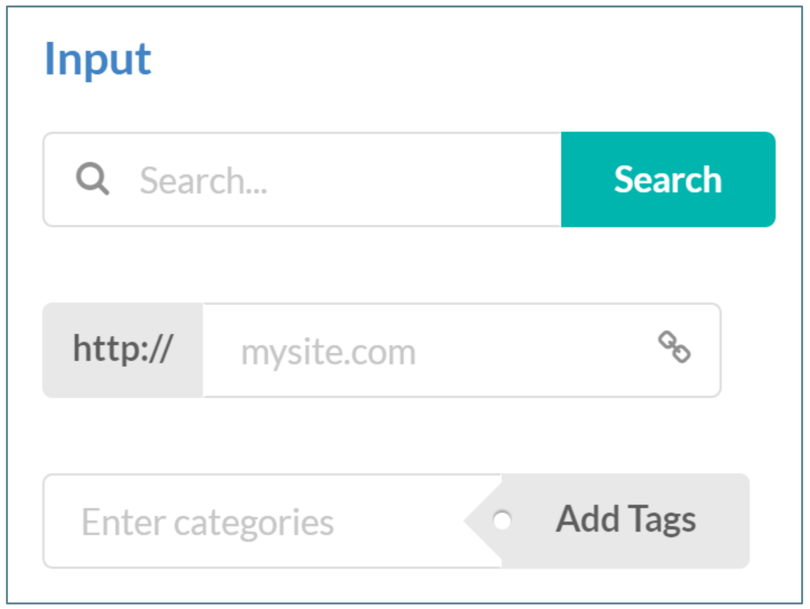
Icon

An icon is a glyph used to represent another concept more simply

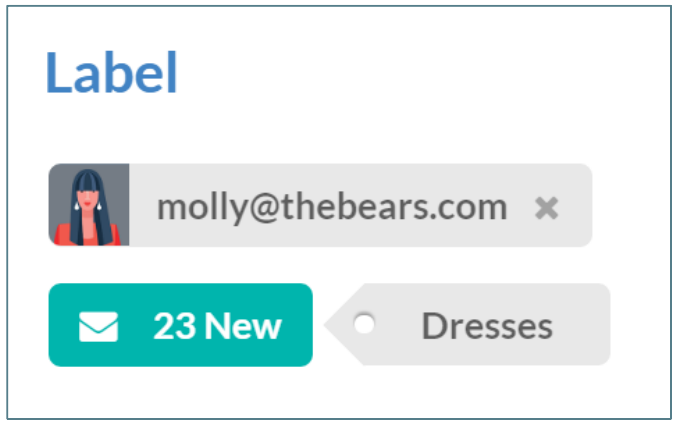


Header

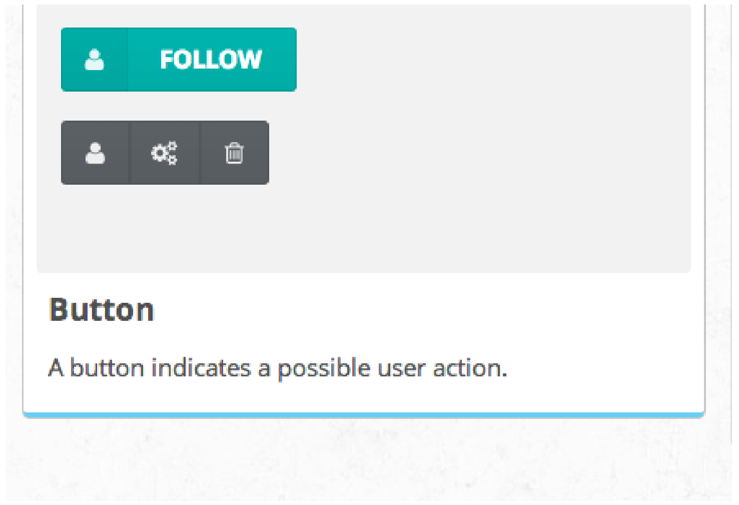
Headers provide a short summary of content



Input

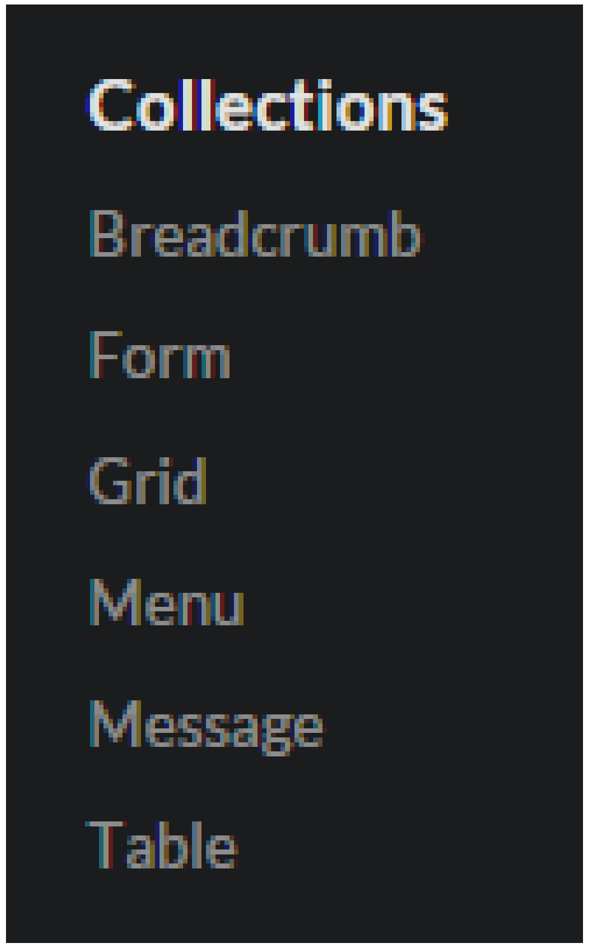


Label



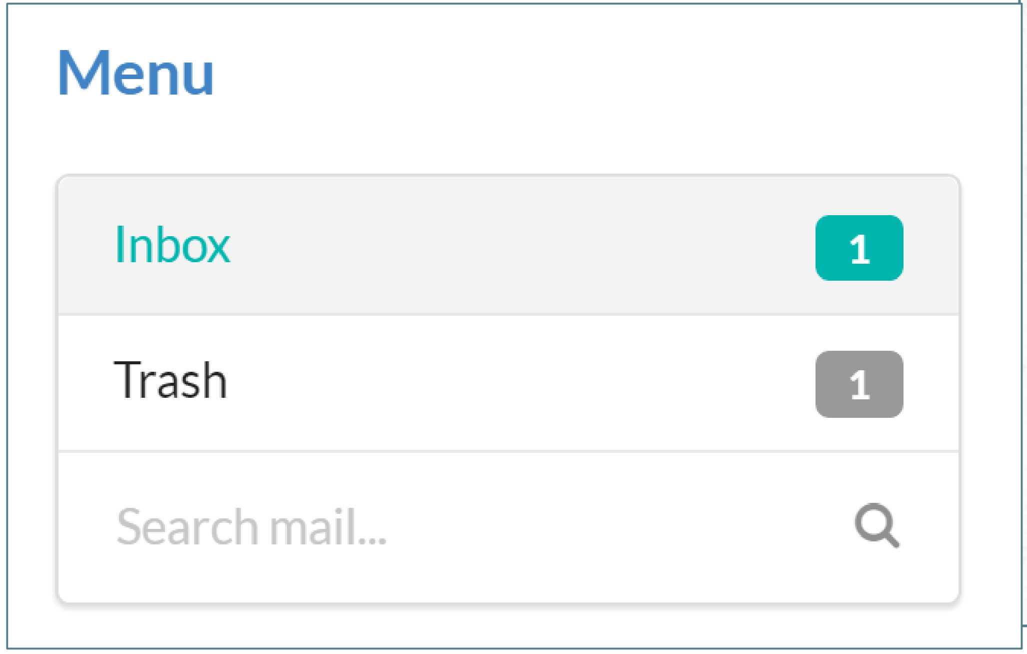
Button

A button indicates a possible user action.



Collections

Collections are heterogeneous groups of components which are usually found together. They describe a list of "usual suspects" which appear in a certain context.

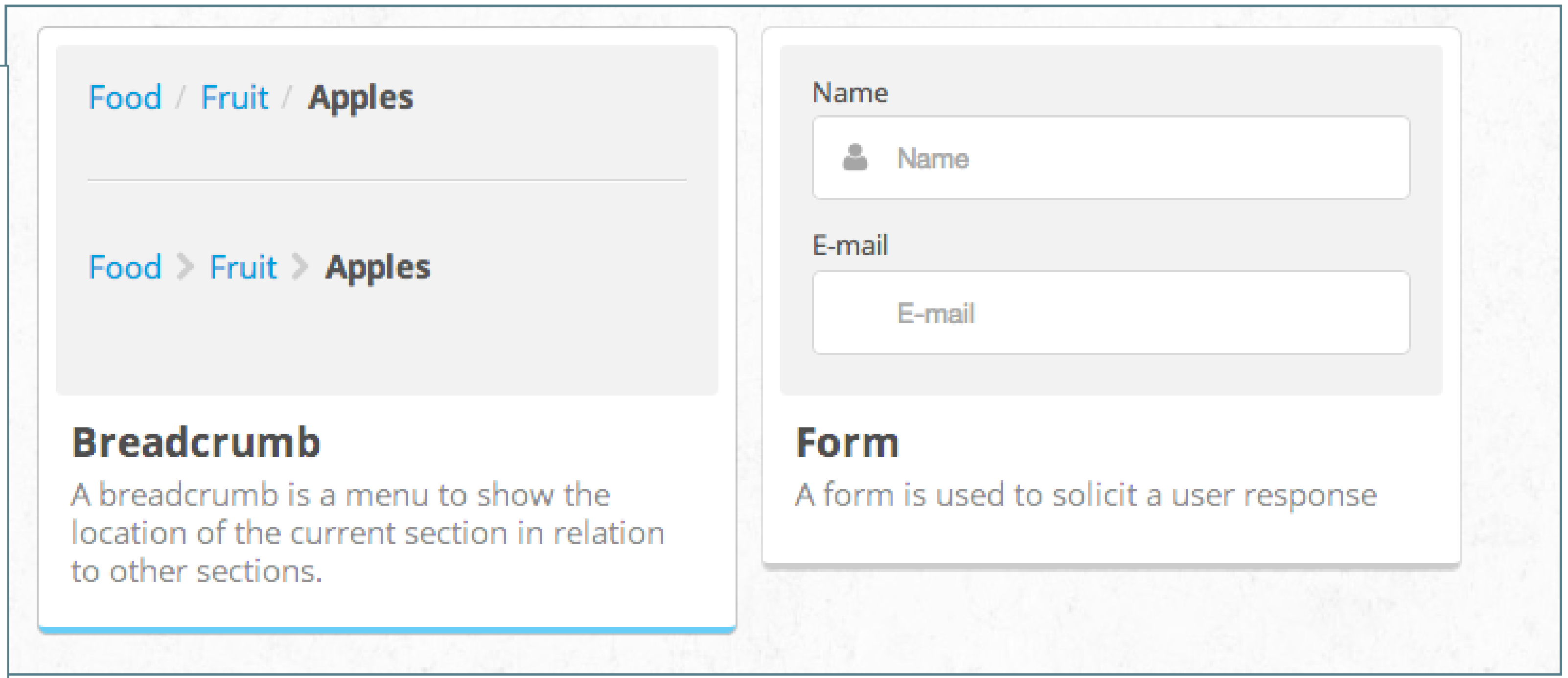


Menu

Inbox 1

Trash 1

Search mail...



Food / Fruit / **Apples**

Food > Fruit > **Apples**

Breadcrumb

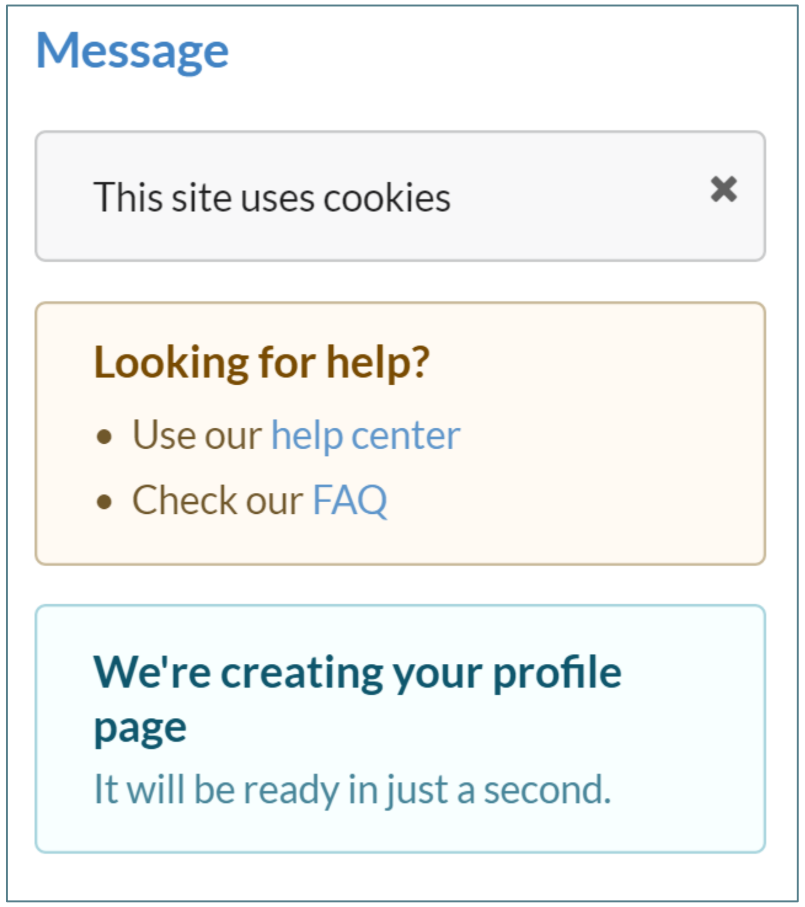
A breadcrumb is a menu to show the location of the current section in relation to other sections.

Name: Name

E-mail: E-mail

Form

A form is used to solicit a user response



Message

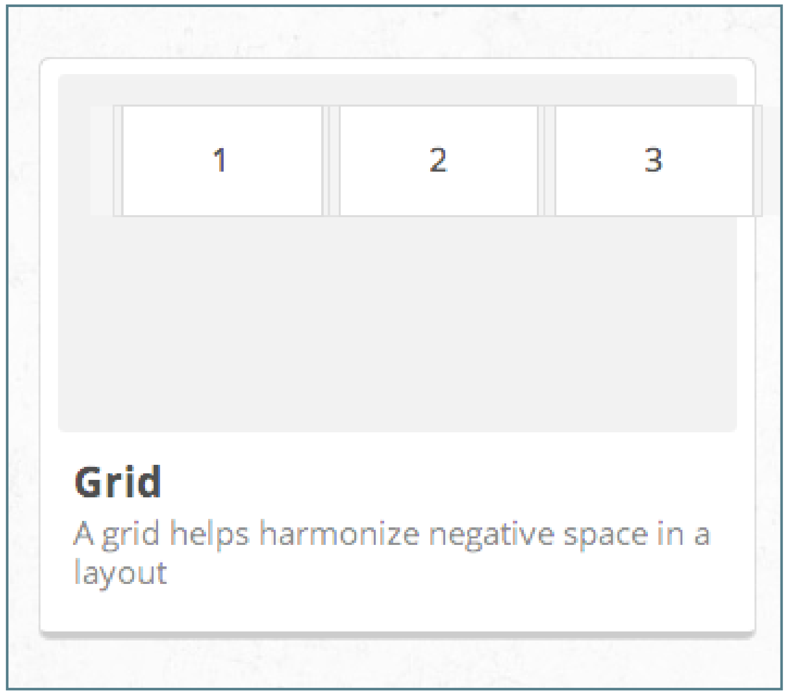
This site uses cookies ✖

Looking for help?

- Use our help center
- Check our FAQ

We're creating your profile page

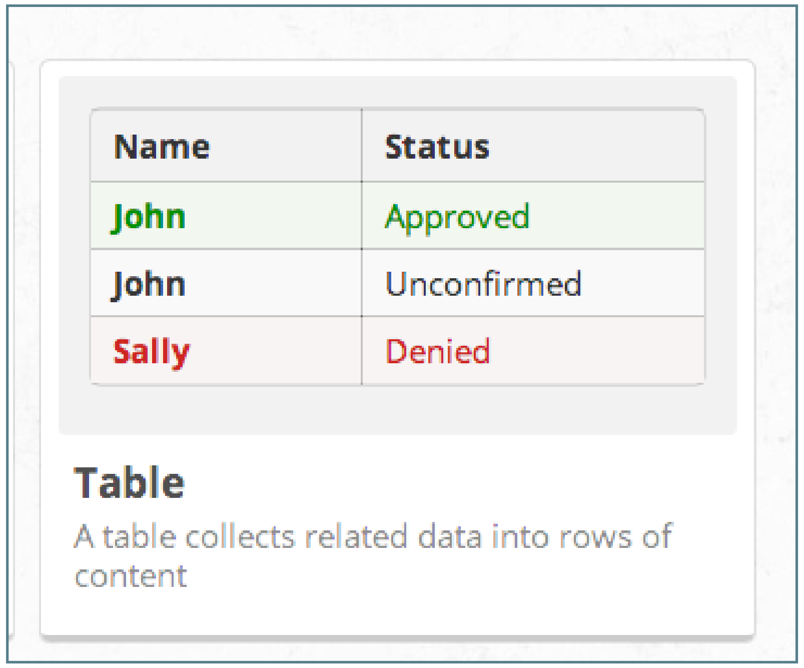
It will be ready in just a second.



1	2	3

Grid

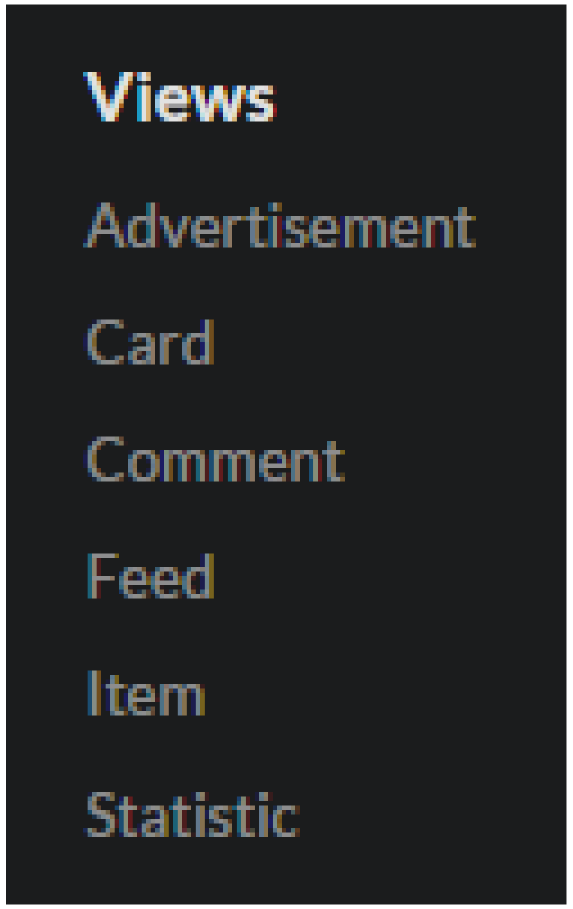
A grid helps harmonize negative space in a layout



Name	Status
John	Approved
John	Unconfirmed
Sally	Denied

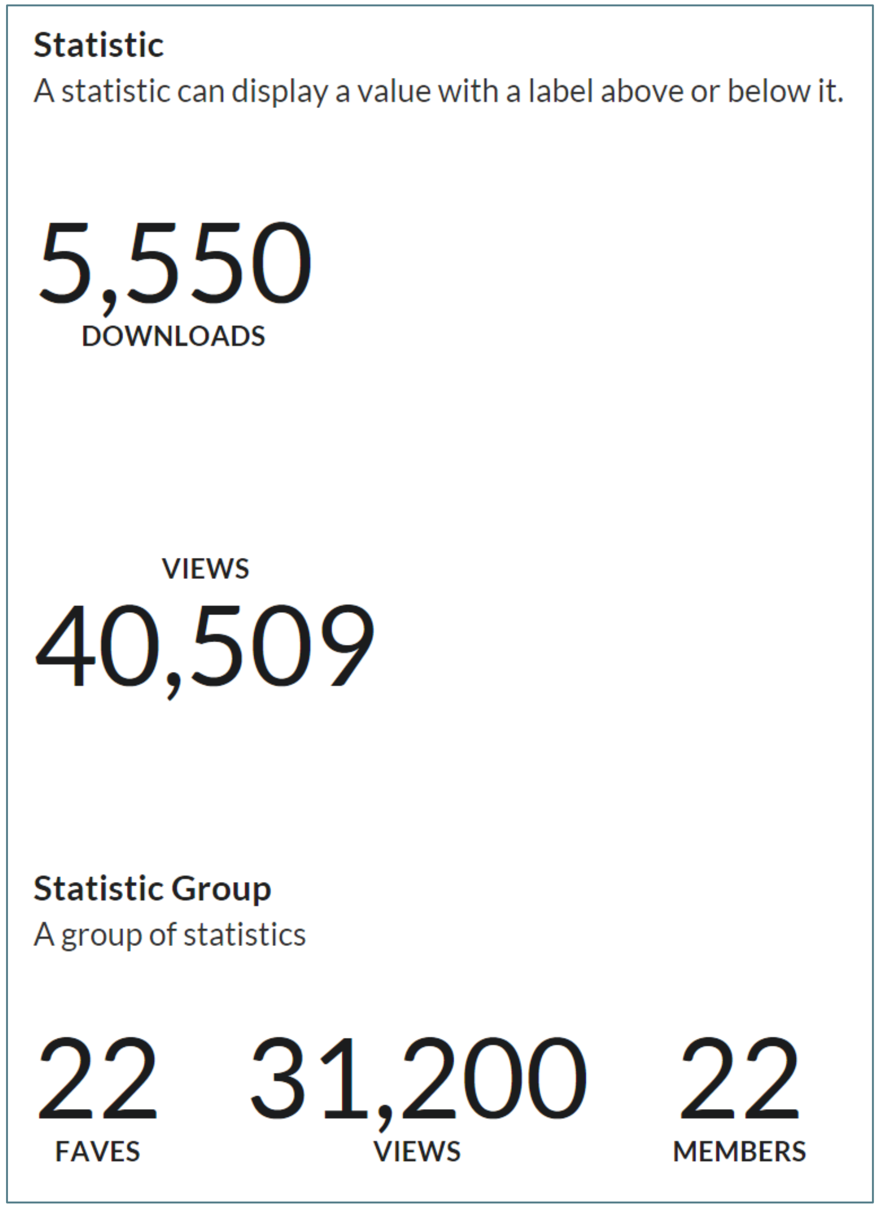
Table

A table collects related data into rows of content



Views

A view is a convention for presenting specific types of content that is usually consistent across a website.



Statistic

A statistic can display a value with a label above or below it.

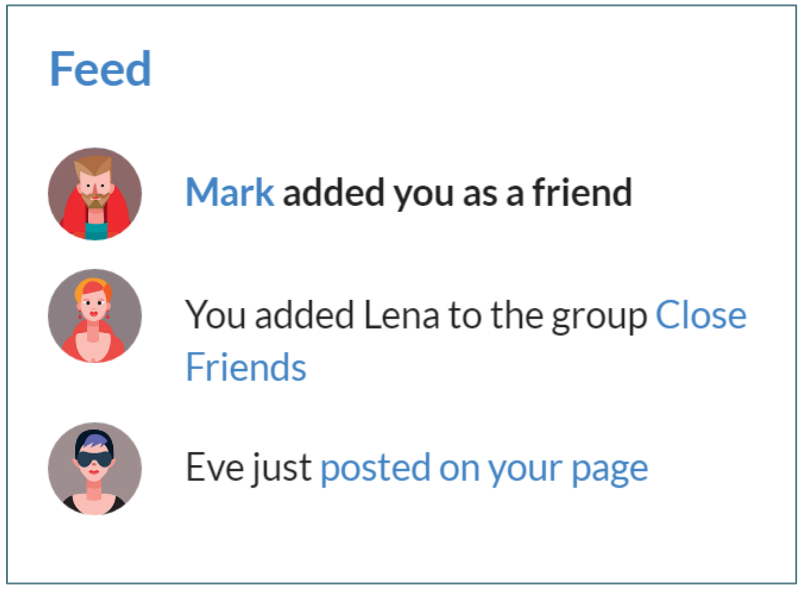
5,550
DOWNLOADS

VIEWS
40,509

Statistic Group

A group of statistics

22	31,200	22
FAVES	VIEWS	MEMBERS

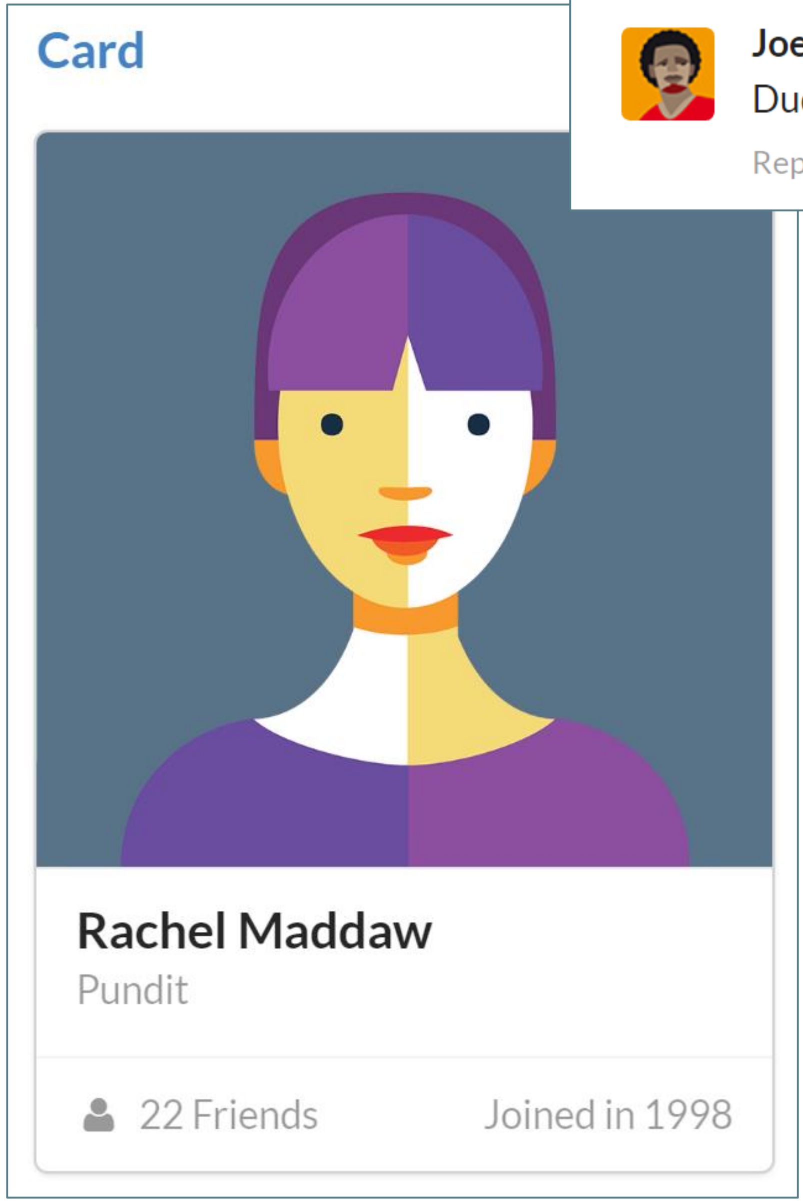


Feed

Mark added you as a friend

You added Lena to the group Close Friends

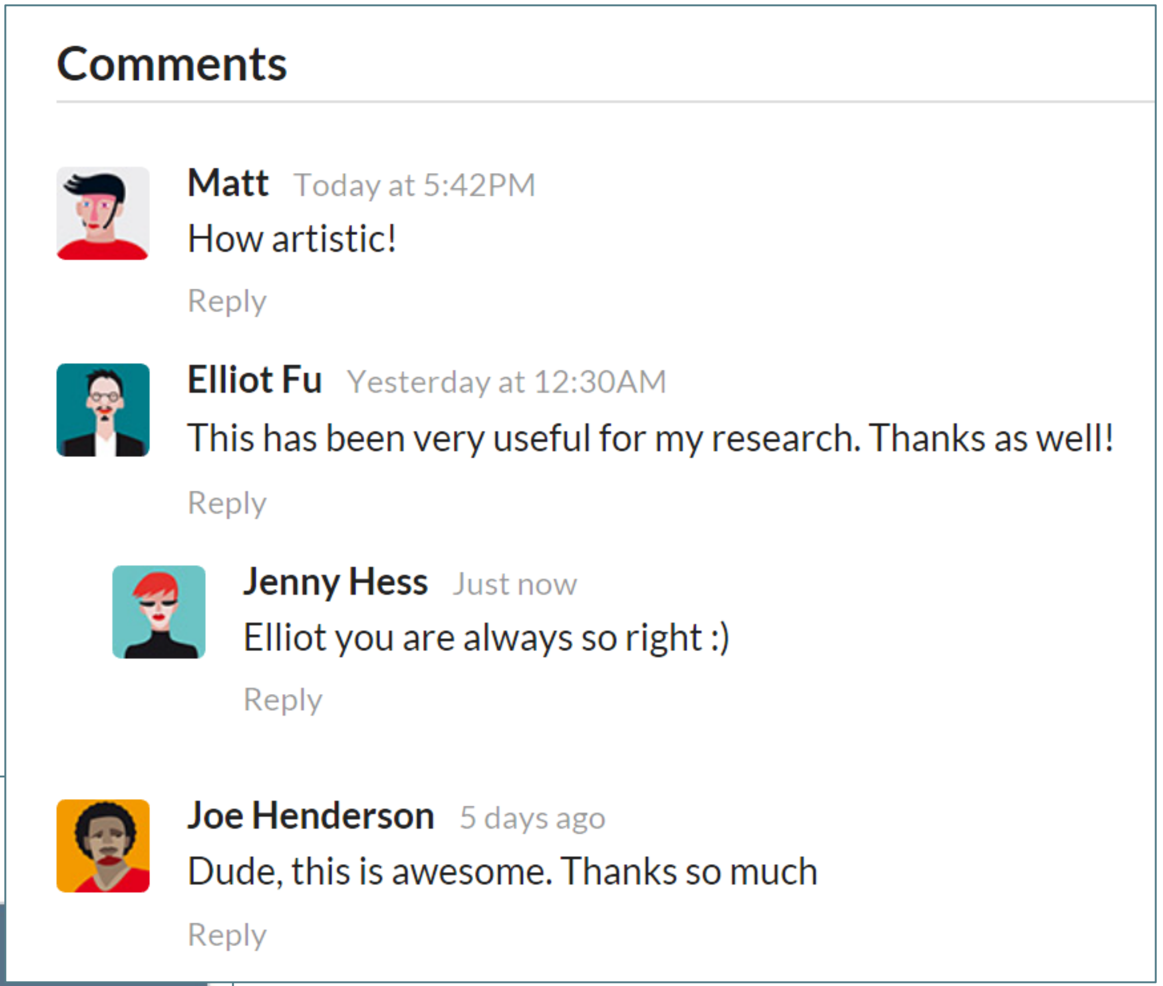
Eve just posted on your page



Card

Rachel Maddaw
Pundit

22 Friends   Joined in 1998



Comments

Matt Today at 5:42PM
How artistic!
Reply

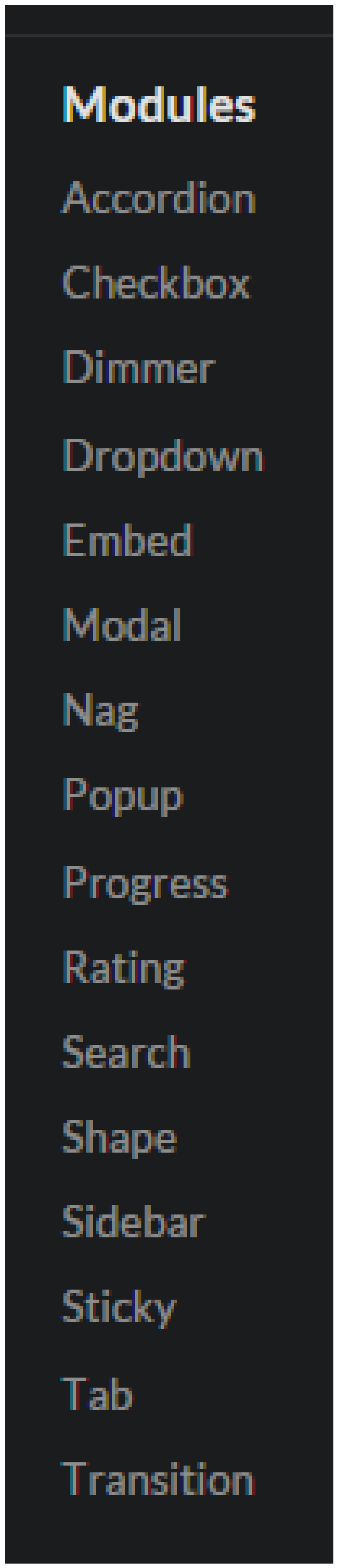
Elliot Fu Yesterday at 12:30AM
This has been very useful for my research. Thanks as well!
Reply

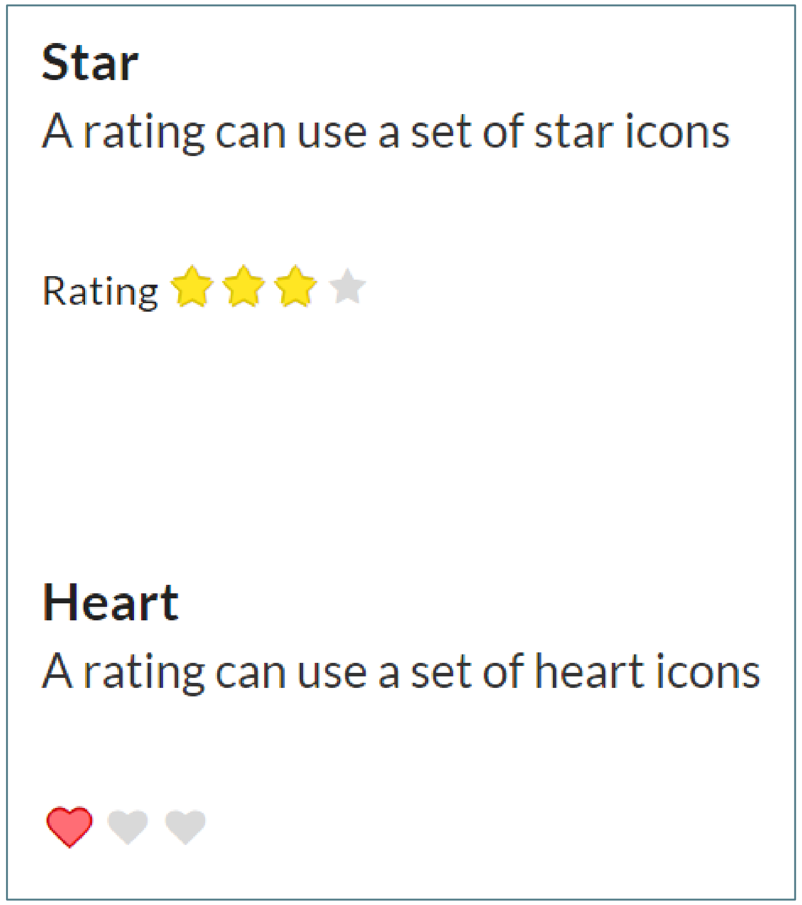
Jenny Hess Just now
Elliot you are always so right :)
Reply

Joe Henderson 5 days ago
Dude, this is awesome. Thanks so much
Reply

Modules

Modules are components that include both a definition of how they appear and how they behave.





Star

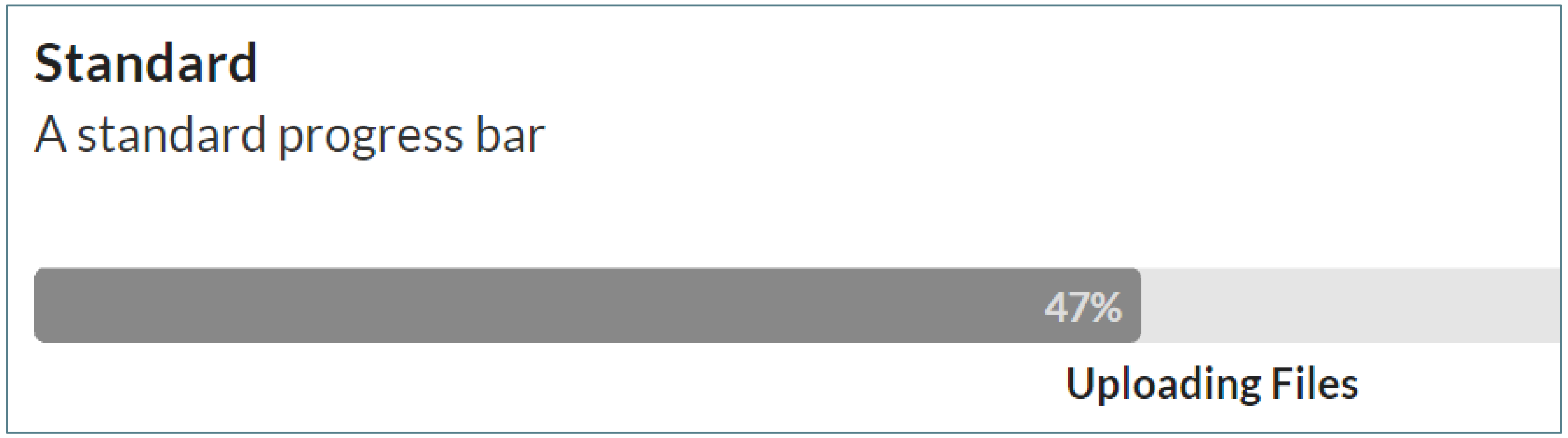
A rating can use a set of star icons

Rating ★★★☆

Heart

A rating can use a set of heart icons

♥♡♡

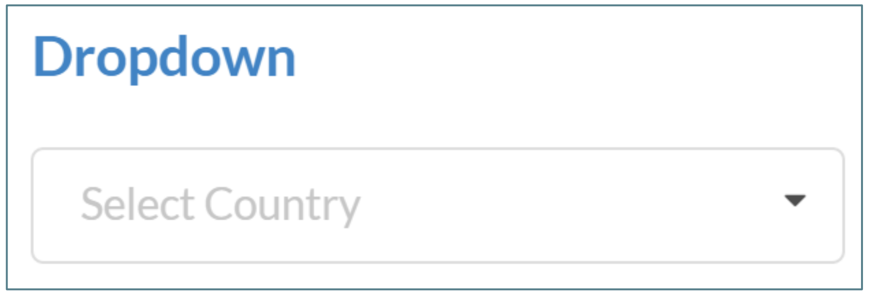


Standard

A standard progress bar

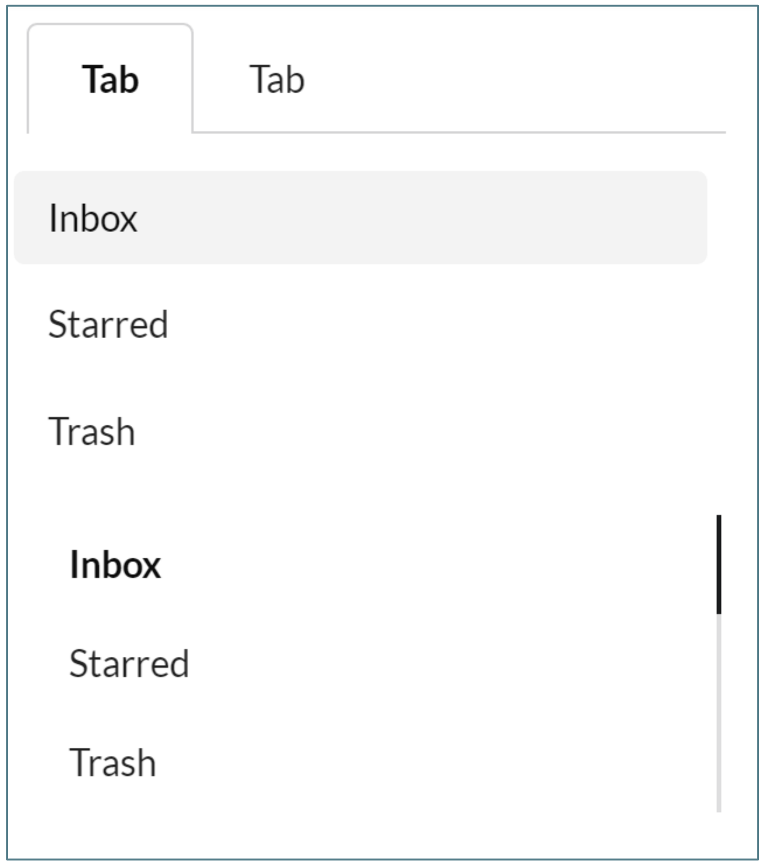
47%

Uploading Files



Dropdown

Select Country ▾



Tab | Tab

Inbox

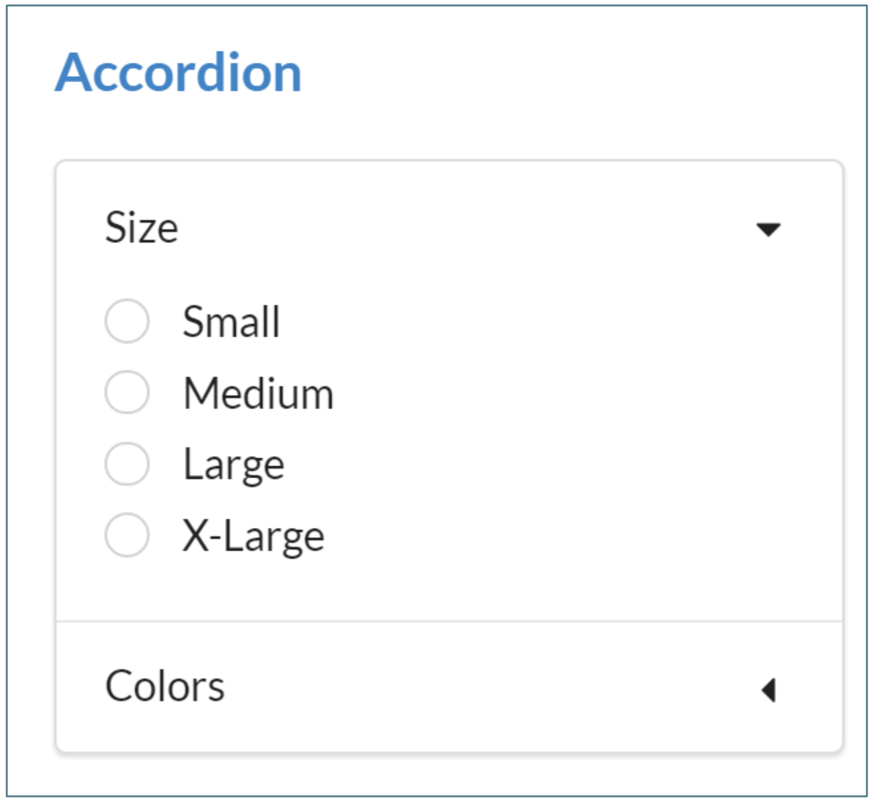
Starred

Trash

Inbox

Starred

Trash



Accordion

Size ▾

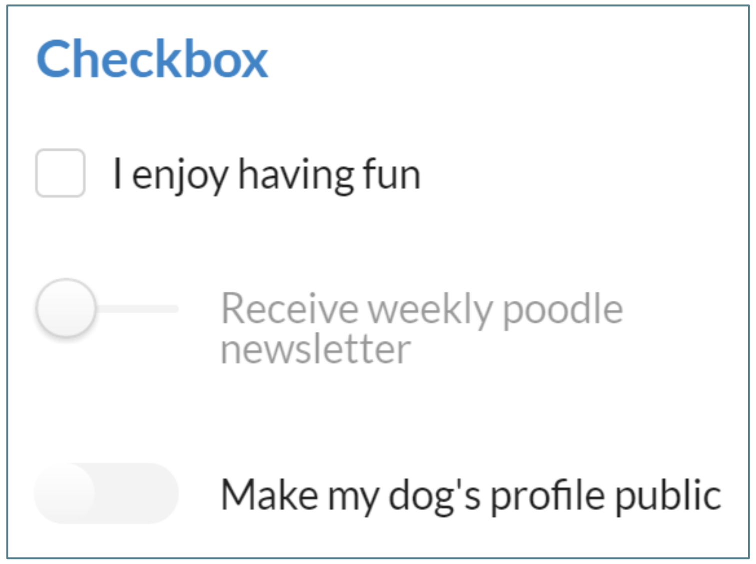
○ Small

○ Medium

○ Large

○ X-Large

Colors ◂

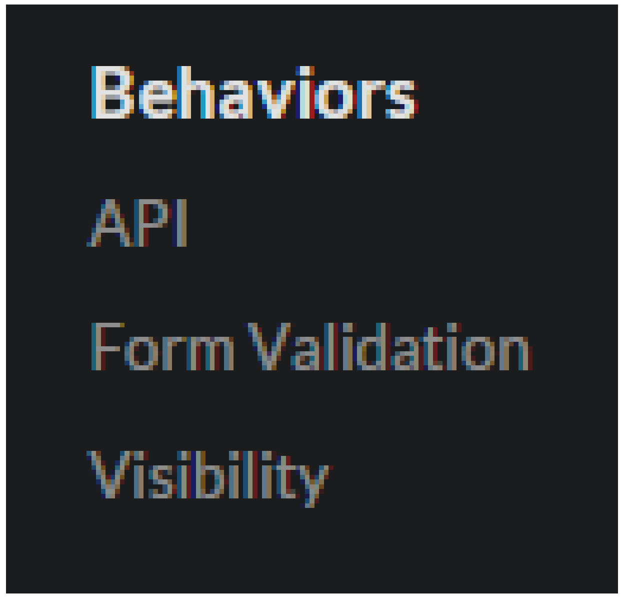


Checkbox

☐ I enjoy having fun

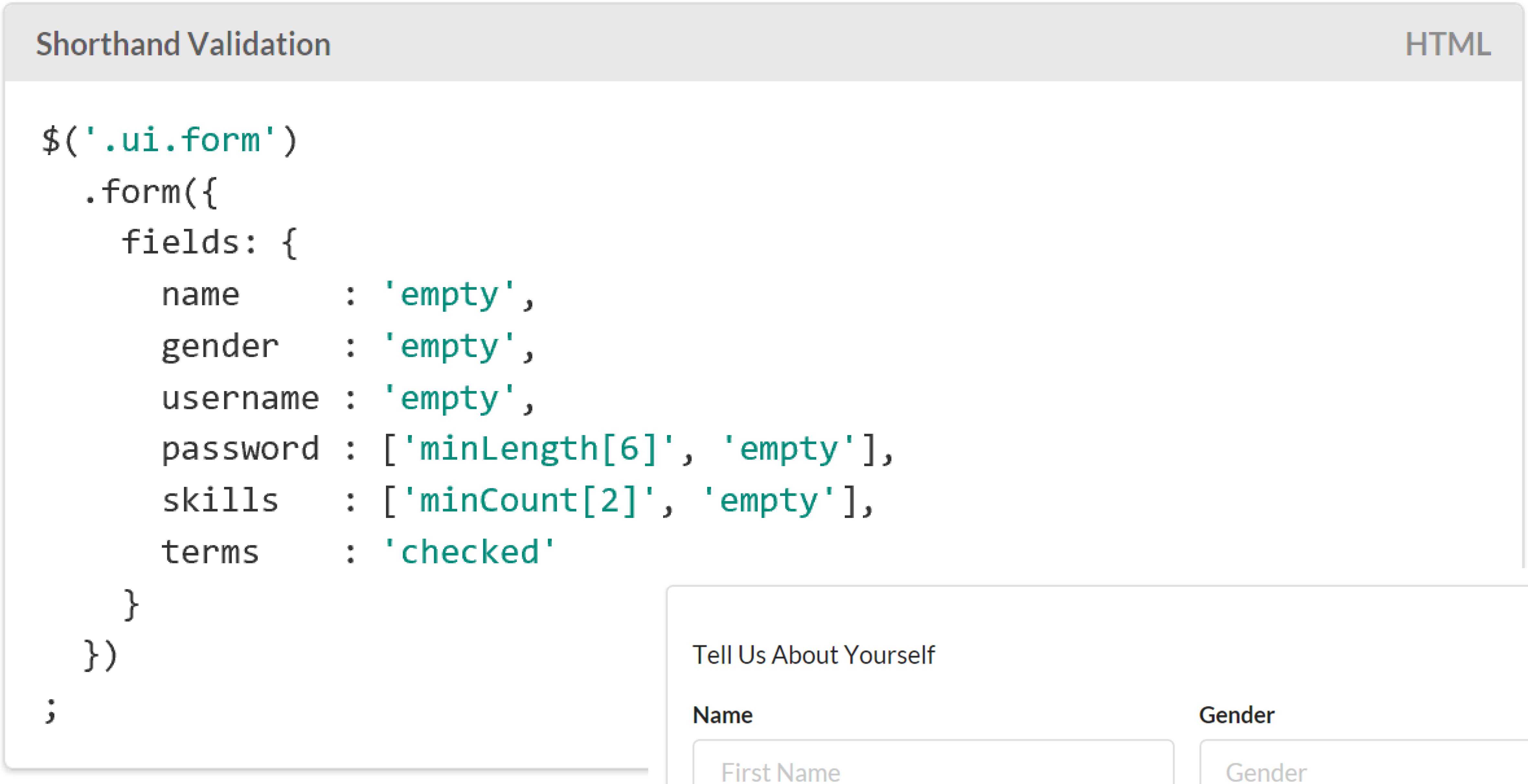
Receive weekly poodle newsletter

Make my dog's profile public

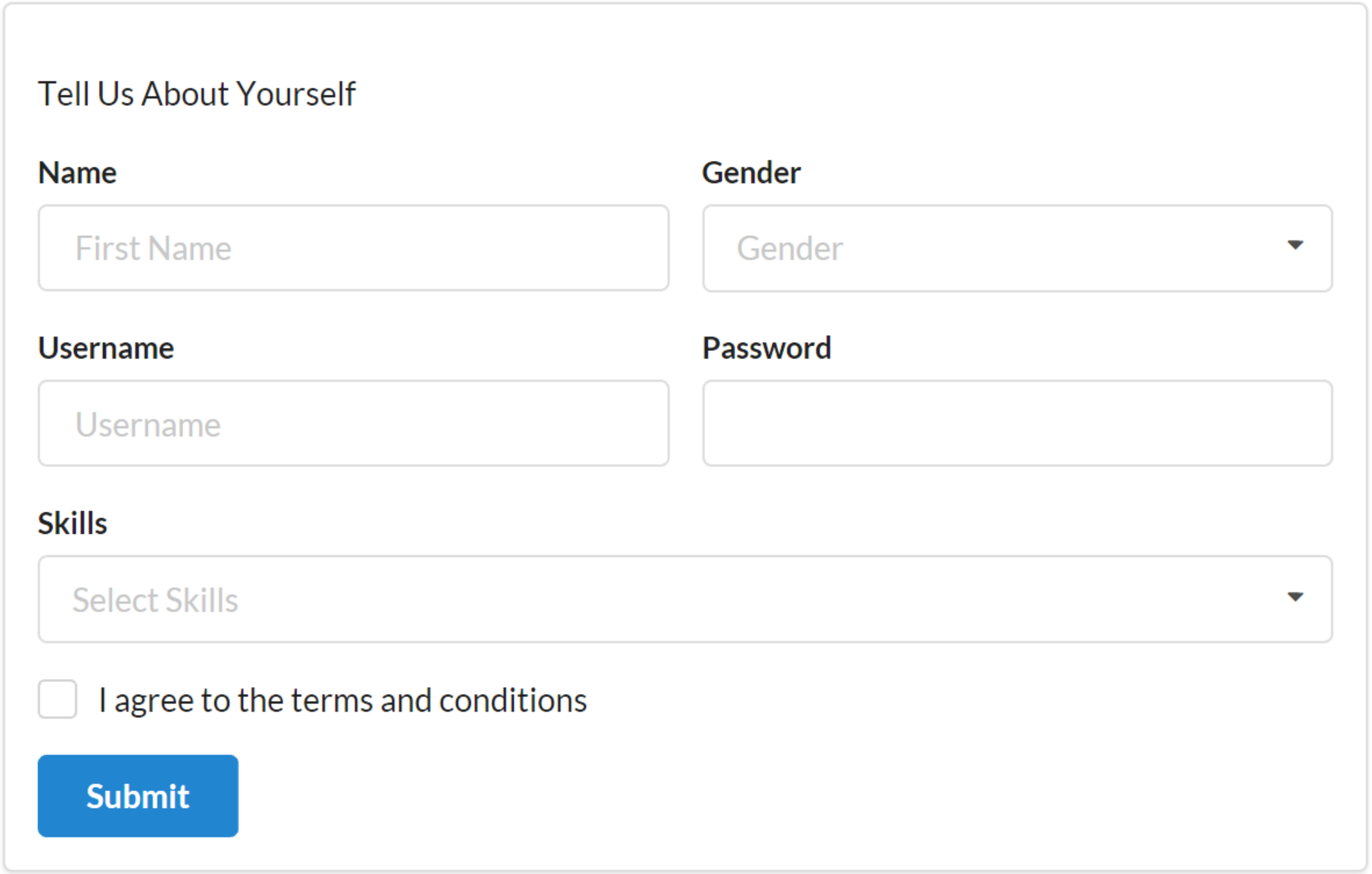


Behaviours

Behaviors are standalone Javascript components that describe how page elements should act, but not how they should appear.


Shorthand Validation HTML

```
$('.ui.form')
  .form({
    fields: {
      name     : 'empty',
      gender   : 'empty',
      username : 'empty',
      password : ['minLength[6]', 'empty'],
      skills   : ['minCount[2]', 'empty'],
      terms    : 'checked'
    }
  })
;
```


Tell Us About Yourself

Name	Gender
First Name	Gender
Username	Password
Username	
Skills	
Select Skills	

☐ I agree to the terms and conditions

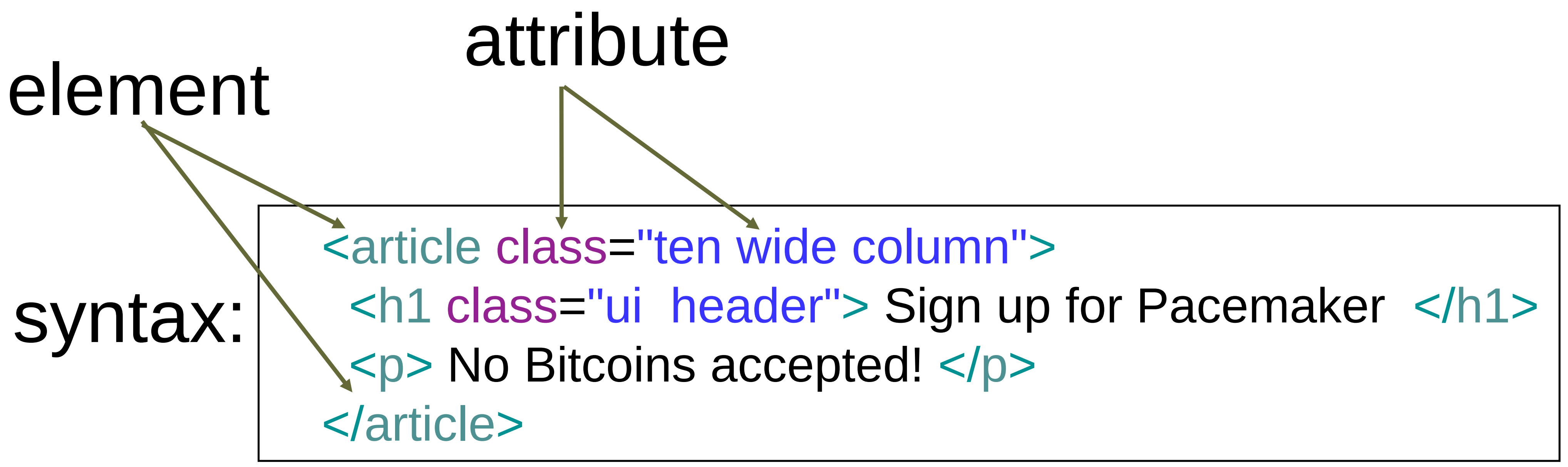
Submit

Topics List

- Overview of Semantic
- Specifics on Semantic
- Installing Semantic with Play
- Using Semantic with Play

metadata	• <html> <head>
sections	• <body><section><article><nav><aside><header><footer><h1><h2><h3>
grouping	• <p><ul> <ol> <li> <div>
links	• <a>
embedding	• <img>
forms	• <form>
tabular data	• <table>

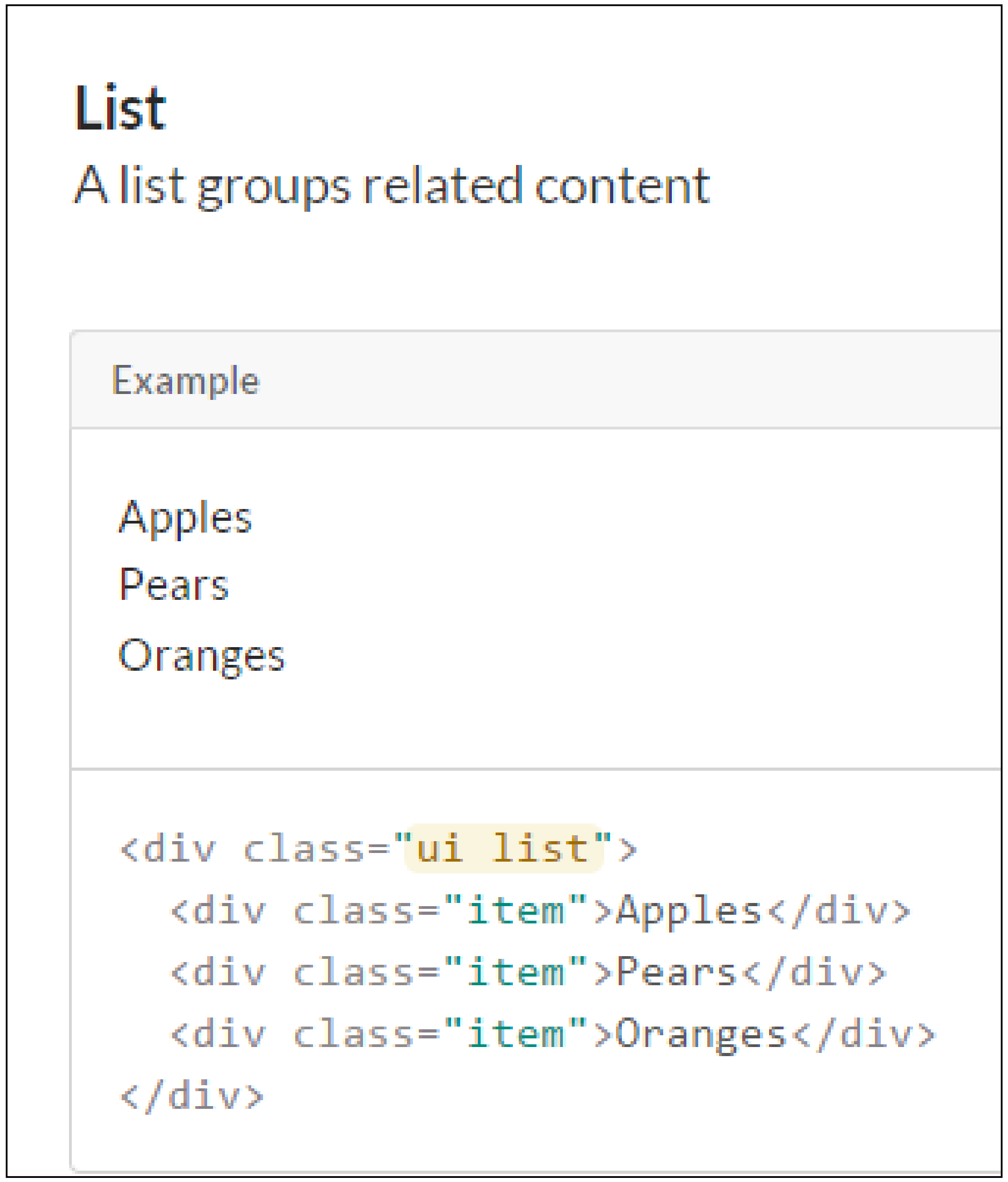
+ attribute syntax:



```
<article class="ten wide column">
  <h1 class="ui  header"> Sign up for Pacemaker  </h1>
  <p> No Bitcoins accepted! </p>
</article>
```

ui – Special Class

- **ui** is a special class name used to distinguish parts of components from the overall component e.g.
  - a list uses the class **ui list** because it has a corresponding definition, however a list item, will just use the class **item**.
- The **ui** class name helps encapsulate CSS rules by making sure all 'parts of a component' are defined in context to a 'whole' component.
- It also helps make scanning unknown code simpler. If you see **ui** you know you are looking at a component.


List

A list groups related content

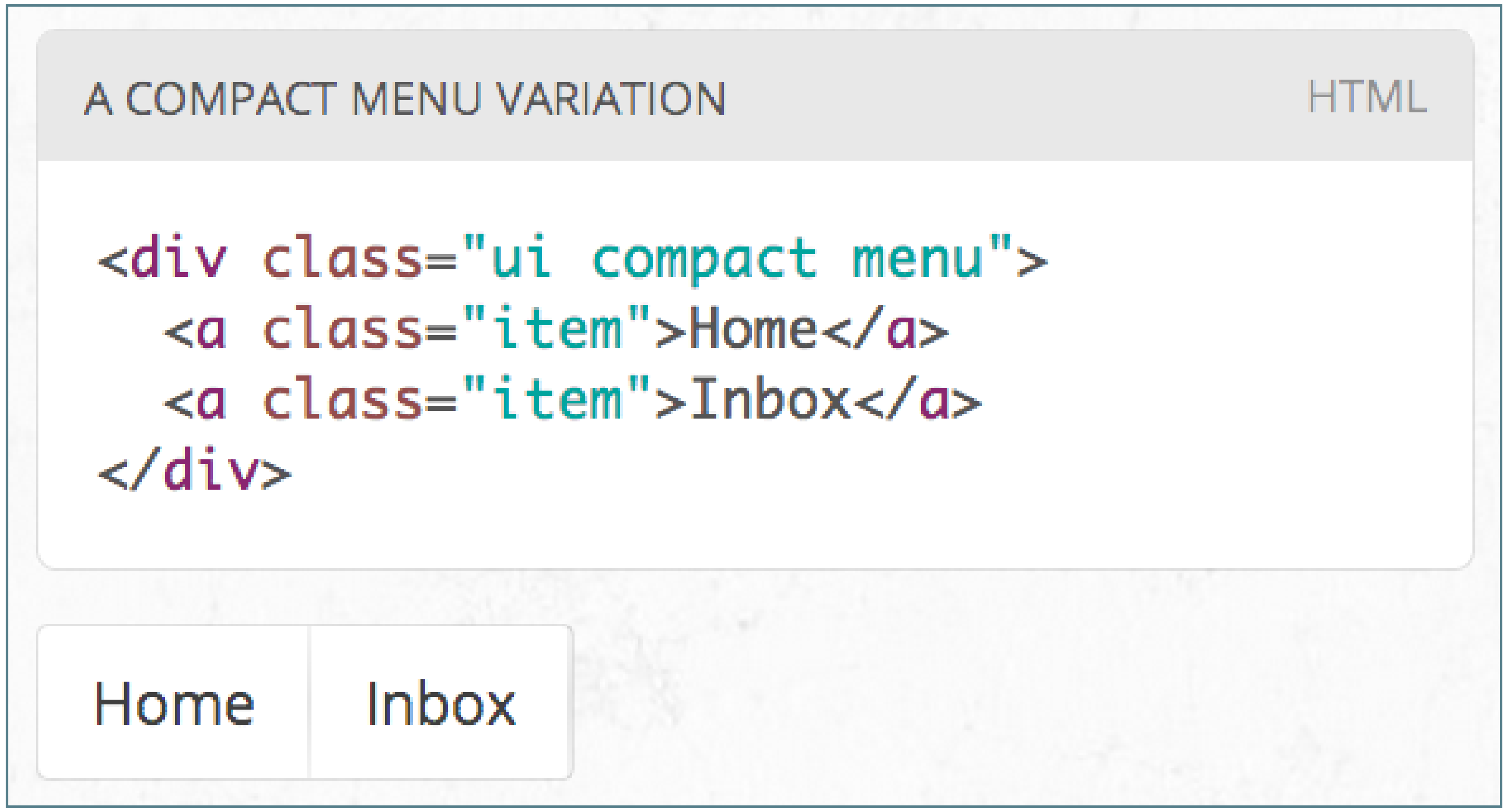
Example

Apples
Pears
Oranges

```
<div class="ui list">
  <div class="item">Apples</div>
  <div class="item">Pears</div>
  <div class="item">Oranges</div>
</div>
```

Changing an Element

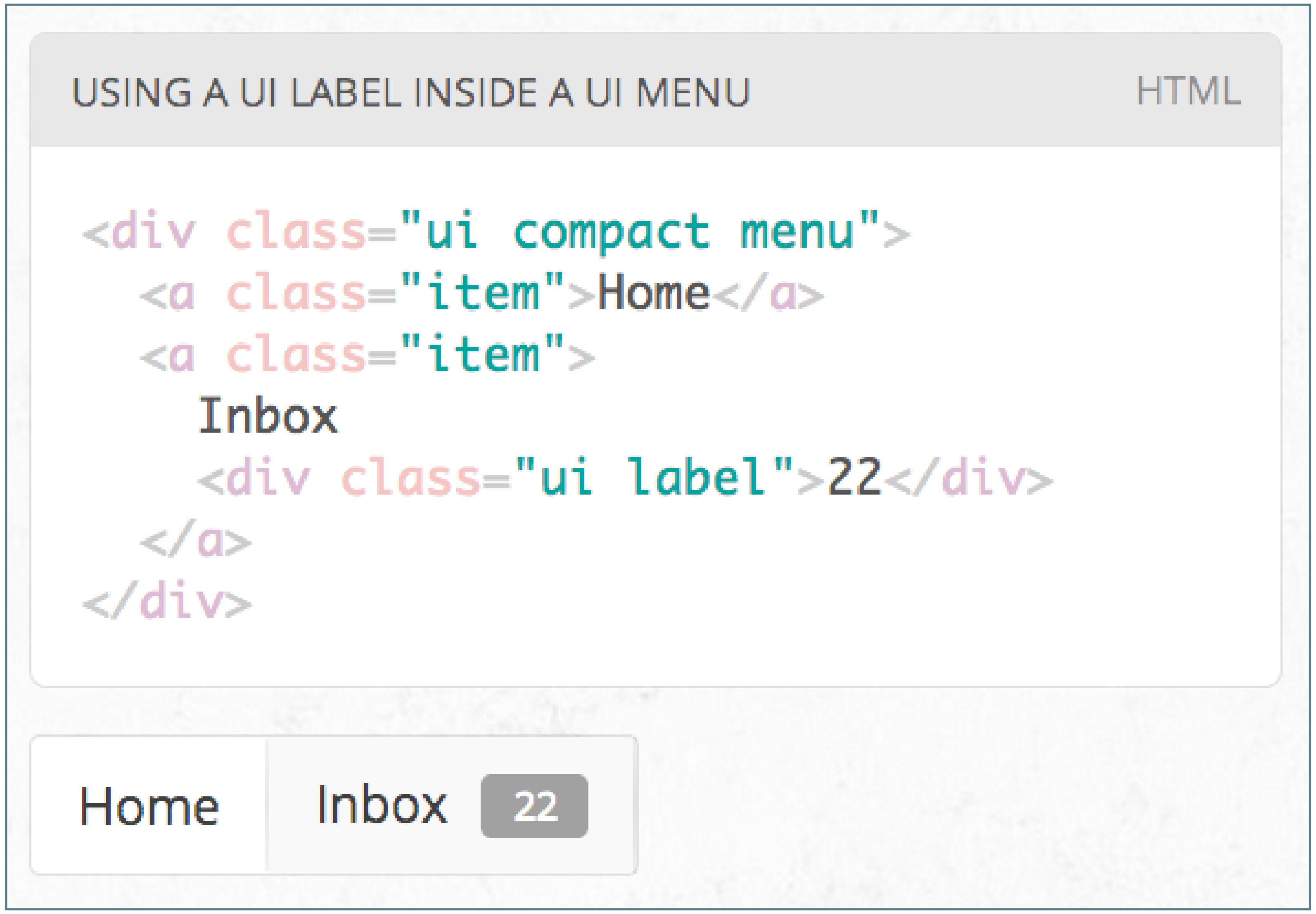
- Class names in Semantic always use single english words.
- If a class name is an adjective it is either a type of element or variation of an element.
- CSS definitions always define adjectives in the context of a noun. In this way class names cannot pollute the namespace.



```html
<div class="ui compact menu">
  <a class="item">Home</a>
  <a class="item">Inbox</a>
</div>
```

Combining an Element

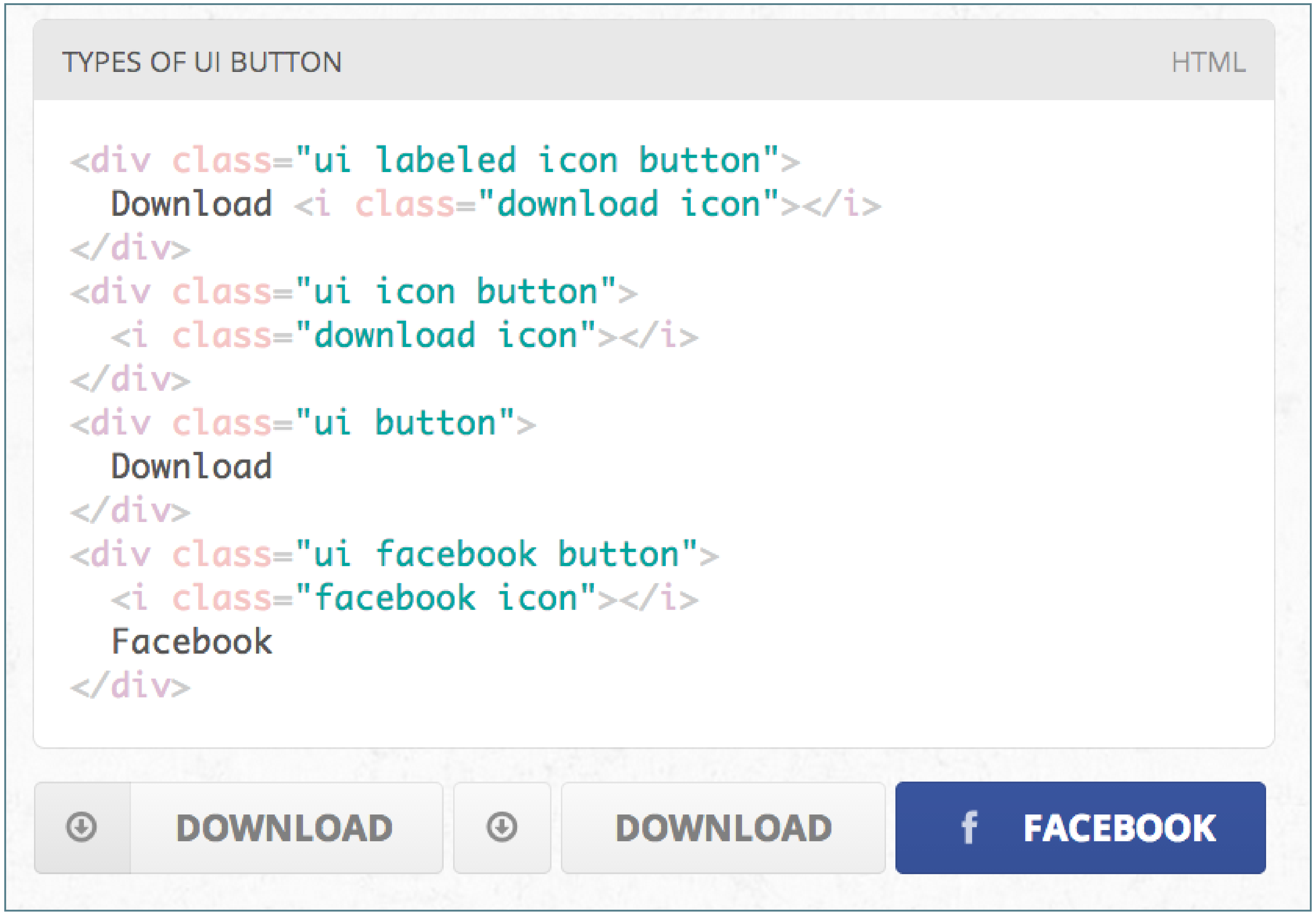
- All UI definitions in semantic are stand-alone, and do not require other components to function.
- However, components can choose to have optional couplings with other components.
- For example you might want to include a badge inside a menu. A label inside of a menu will automatically function as a badge


USING A UI LABEL INSIDE A UI MENU HTML

```
<div class="ui compact menu">
  <a class="item">Home</a>
  <a class="item">
    Inbox
    <div class="ui label">22</div>
  </a>
</div>
```

Types / Variations

- A ui definition in Semantic usually contains a list of mutually exclusive variations on an element design.
- A type is designated by an additional class name on a UI element



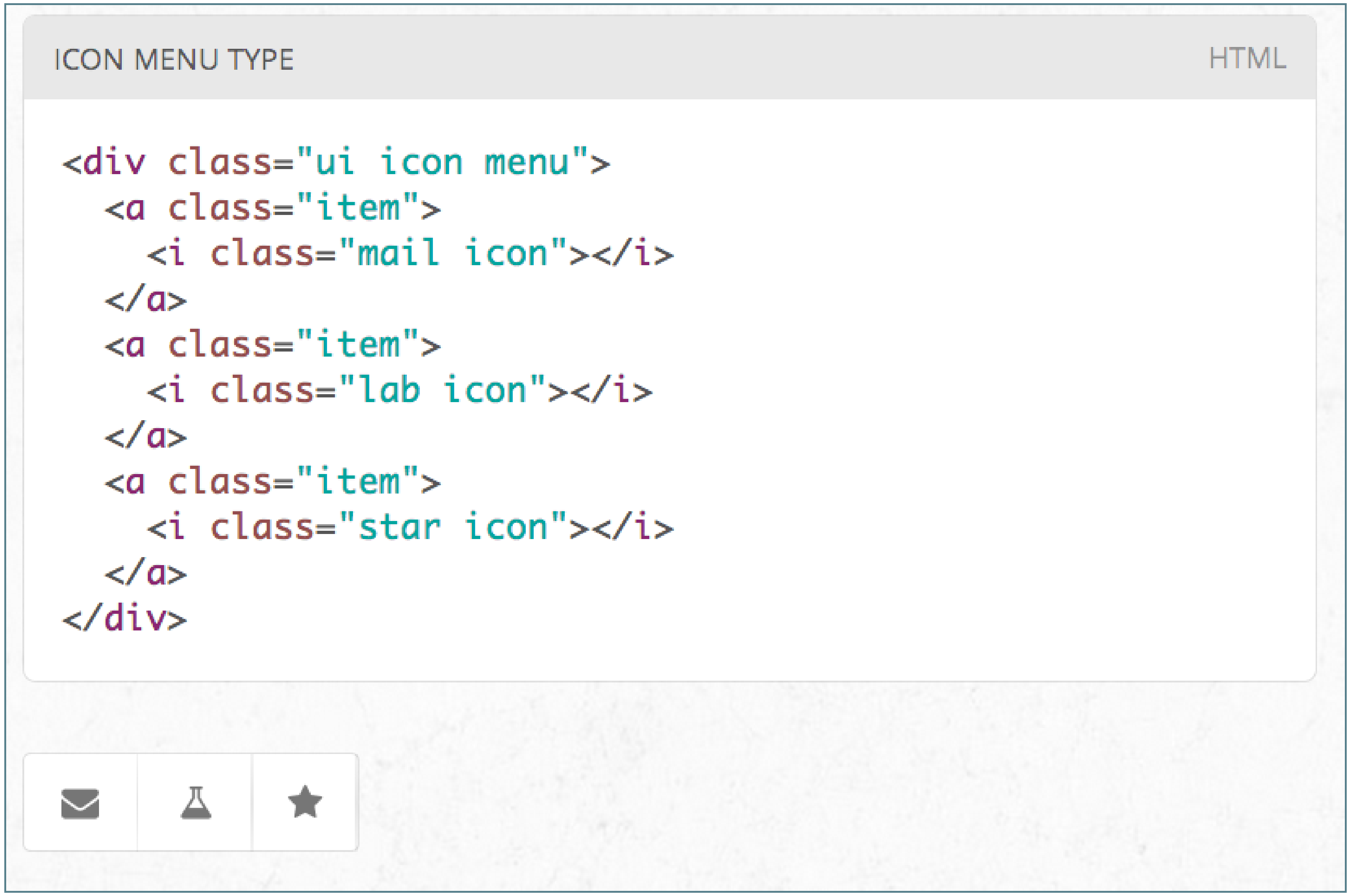
TYPES OF UI BUTTON HTML

```
<div class="ui labeled icon button">
  Download <i class="download icon"></i>
</div>
<div class="ui icon button">
  <i class="download icon"></i>
</div>
<div class="ui button">
  Download
</div>
<div class="ui facebook button">
  <i class="facebook icon"></i>
  Facebook
</div>
```

Types / Content

- Types may require different html structures to work correctly.

- For example, an icon menu might expect different content like icons glyphs instead of text to be formatted correctly

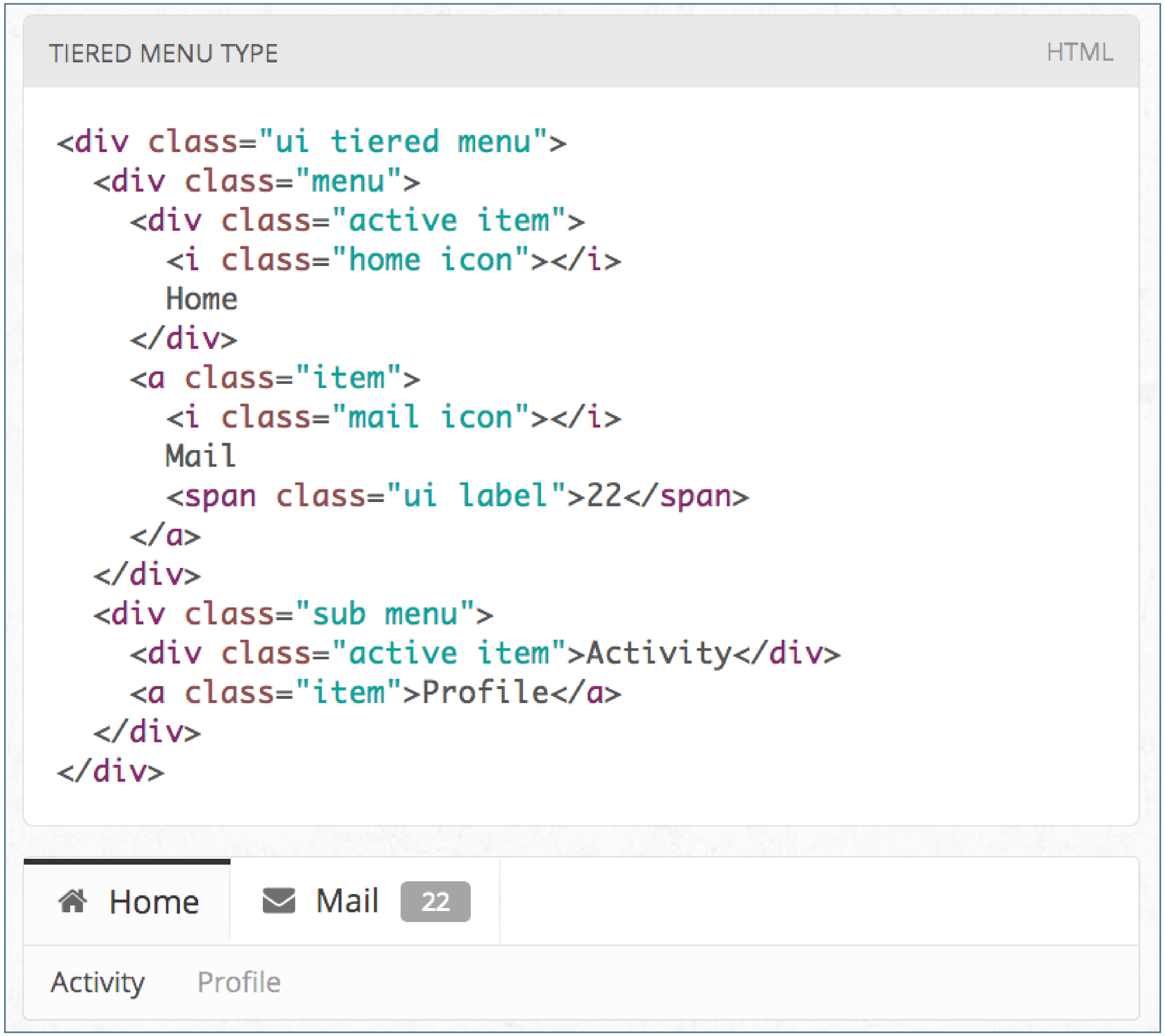


ICON MENU TYPE HTML

```
<div class="ui icon menu">
  <a class="item">
    <i class="mail icon"></i>
  </a>
  <a class="item">
    <i class="lab icon"></i>
  </a>
  <a class="item">
    <i class="star icon"></i>
  </a>
</div>
```

Types / HTML Differences

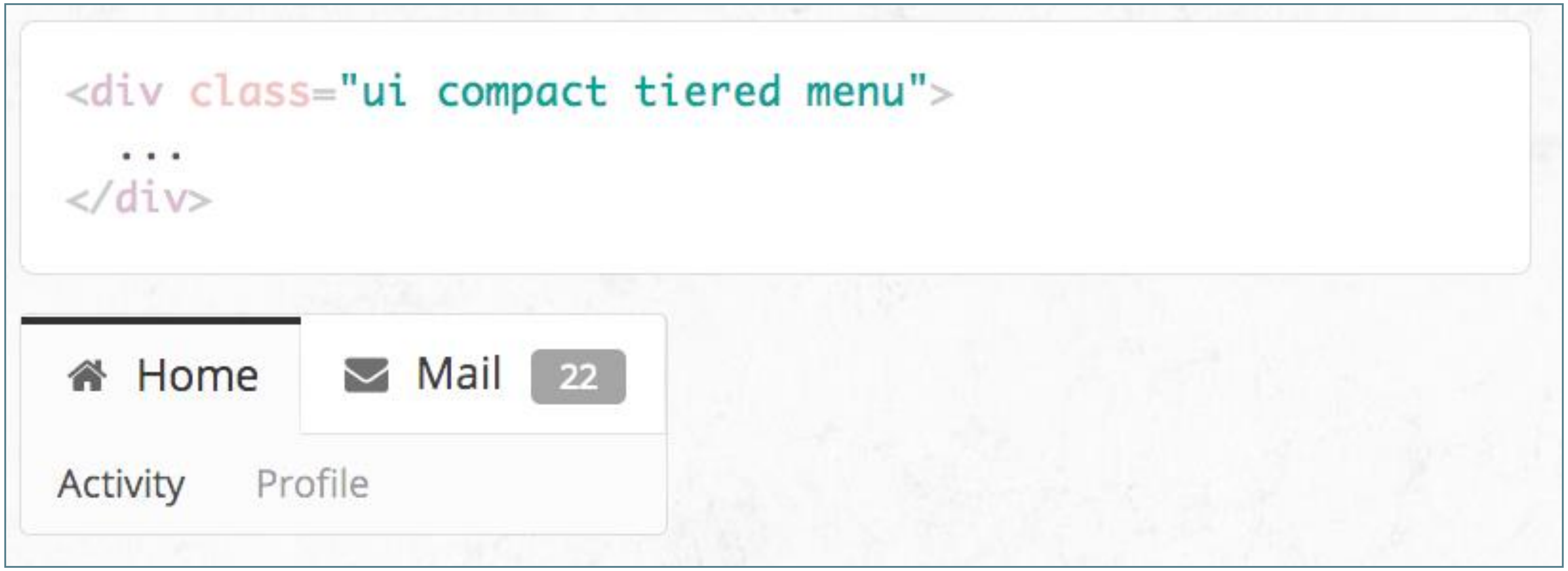
- Types may also each require slightly different html.
- For example, a tiered menu needs html specified for a sub menu to display itself correctly


TIERED MENU TYPE HTML

```
<div class="ui tiered menu">
  <div class="menu">
    <div class="active item">
      <i class="home icon"></i>
      Home
    </div>
    <a class="item">
      <i class="mail icon"></i>
      Mail
      <span class="ui label">22</span>
    </a>
  </div>
  <div class="sub menu">
    <div class="active item">Activity</div>
    <a class="item">Profile</a>
  </div>
</div>
```

Variations

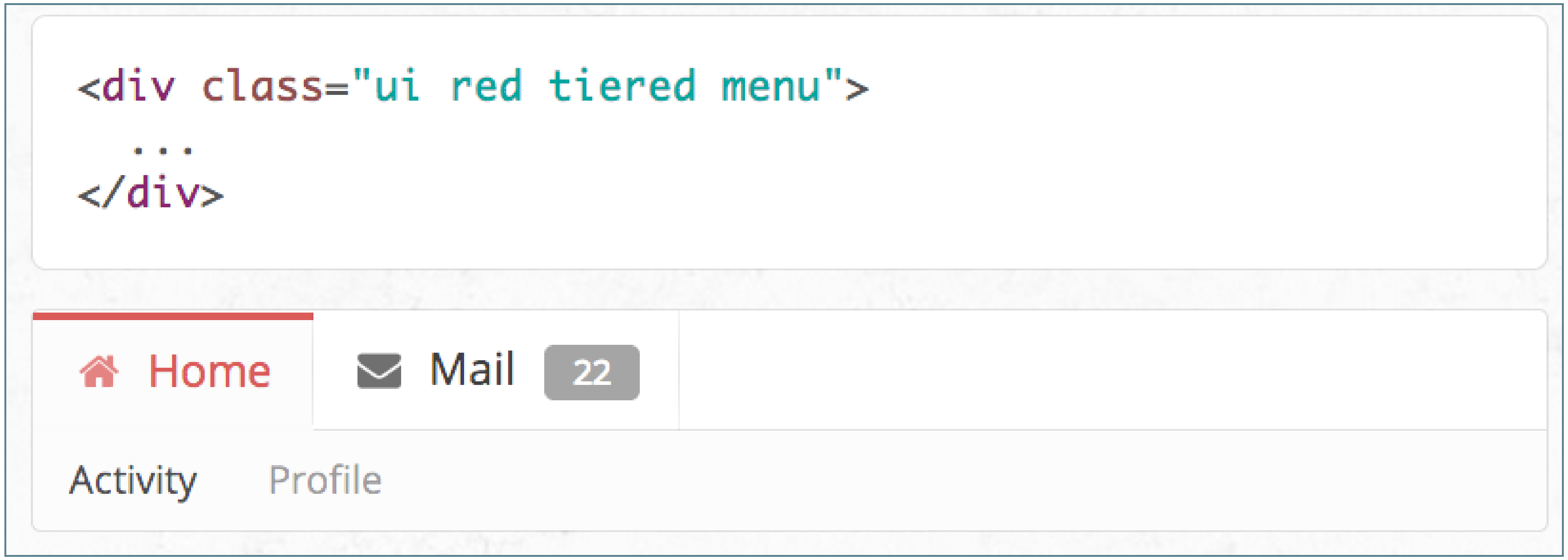
- A variation alters the design of an element but is not mutually exclusive.
- Variations can be stacked together, or be used along with altering an element's type.
- For example, having wide menus that take up the full width of its parent may sometimes be overwhelming. You can use the compact variation of a menu to alter its format to only take up the necessary space.



```
<div class="ui compact tiered menu">
  ...
</div>
```

Intersecting Variations

- The definition for the variation red contains css specifically for describing the intersection of both red and inverted.



```
<div class="ui red tiered menu">
  ...
</div>
```

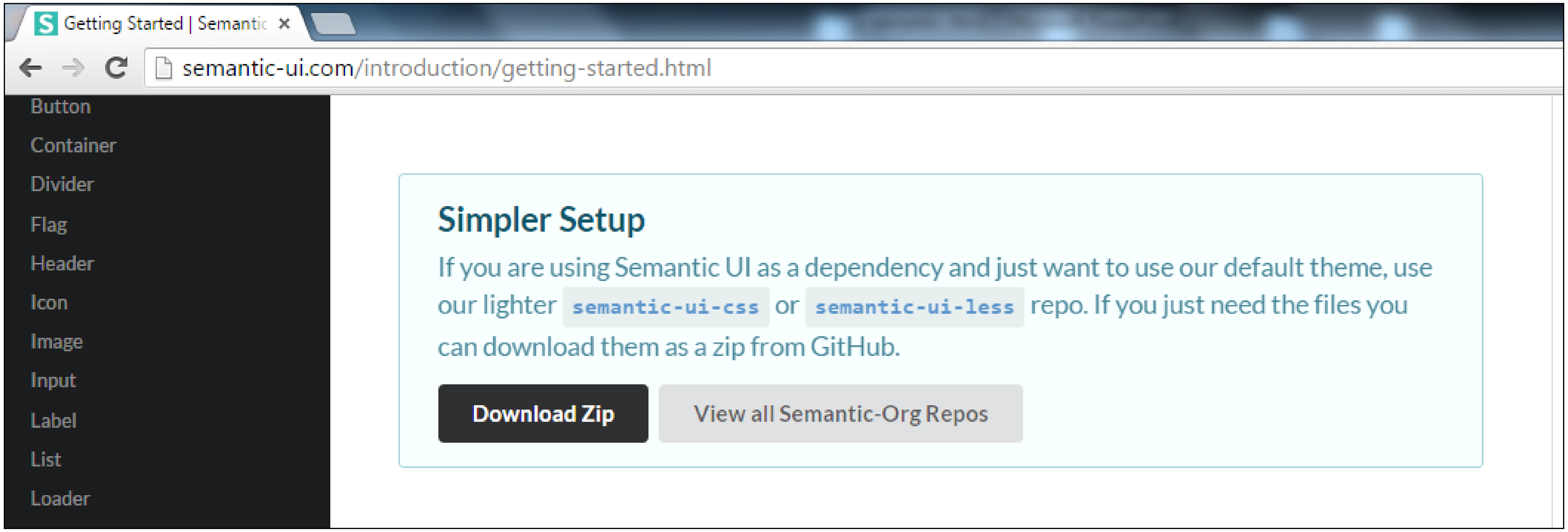
Topics List

- Overview of Semantic
- Specifics on Semantic
- Installing Semantic with Play
- Using Semantic with Play

Semantic UI

http://semantic-ui.com/

Installing Semantic UI as a project dependency

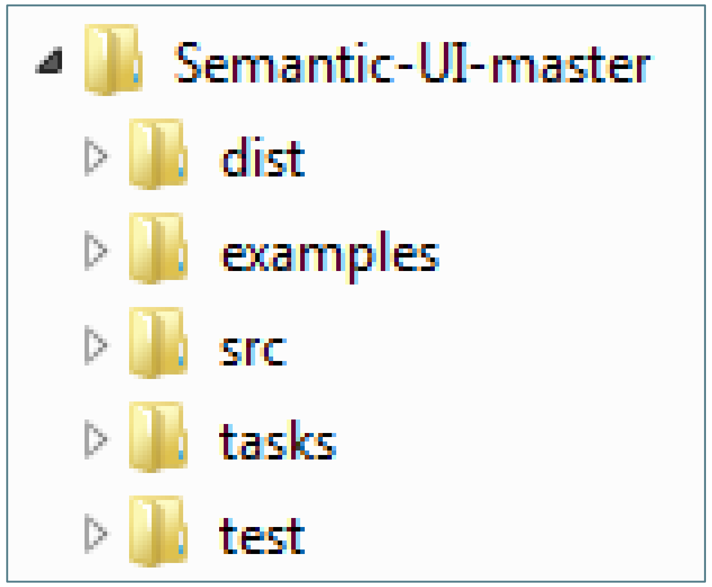

Getting Started | Semantic
semantic-ui.com/introduction/getting-started.html
Button
Container
Divider
Flag
Header
Icon
Image
Input
Label
List
Loader

Simpler Setup

If you are using Semantic UI as a dependency and just want to use our default theme, use our lighter `semantic-ui-css` or `semantic-ui-less` repo. If you just need the files you can download them as a zip from GitHub.

Download Zip
View all Semantic-Org Repos

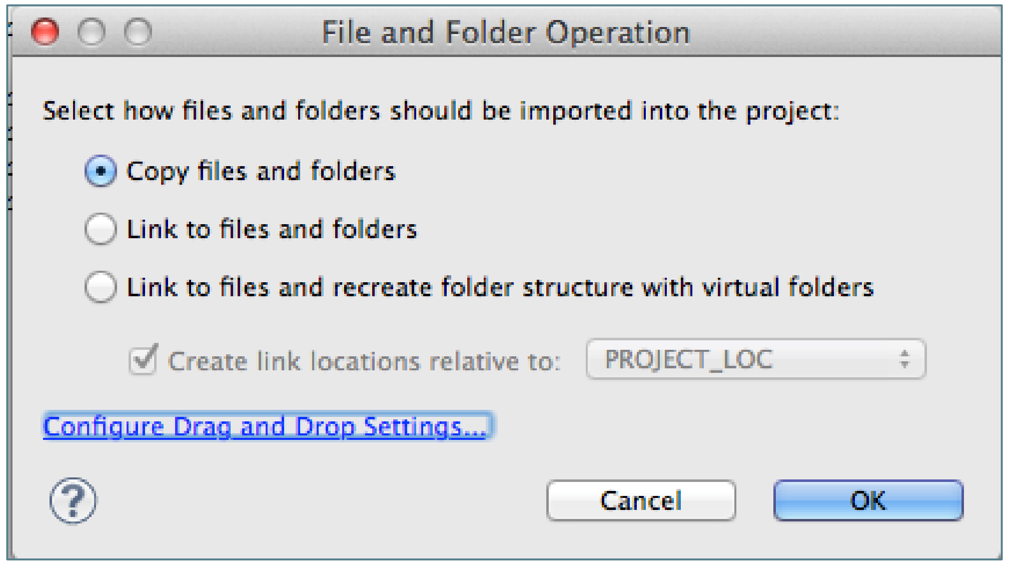
- Download and expand the zip file.
- You should have this file structure →



```
Semantic-UI-master
    dist
    examples
    src
    tasks
    test
```

Installing Semantic UI as a project dependency

- Open your eclipse project and drag the **dist** folder and drop it into the **public** folder.
- When prompted, select the copy files and folders option:



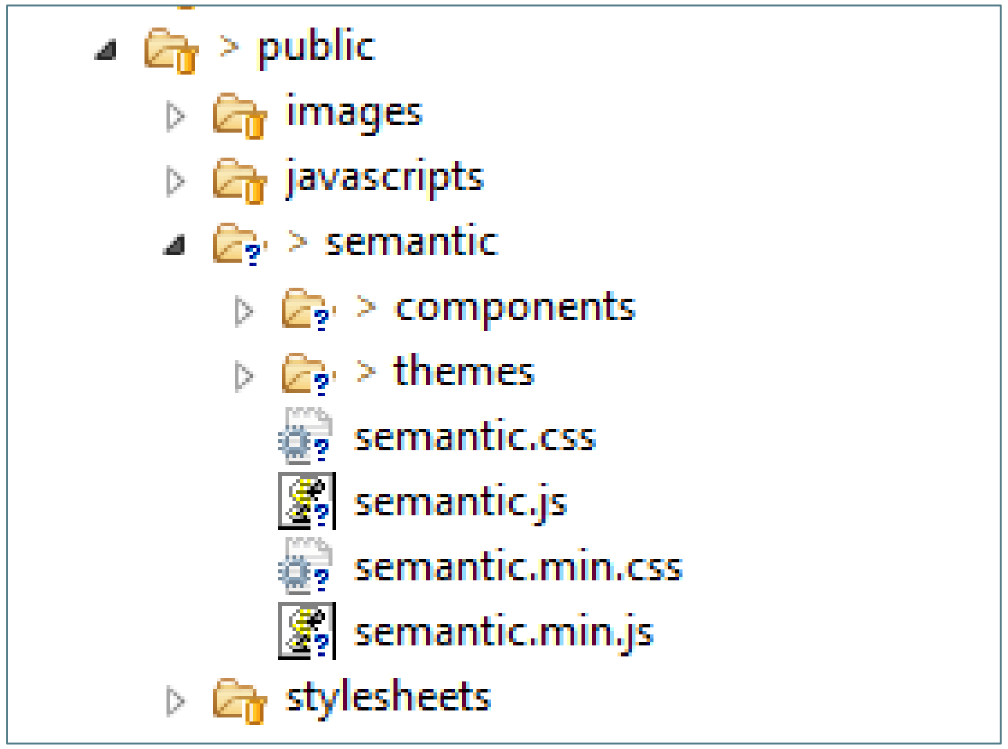
File and Folder Operation

Select how files and folders should be imported into the project:

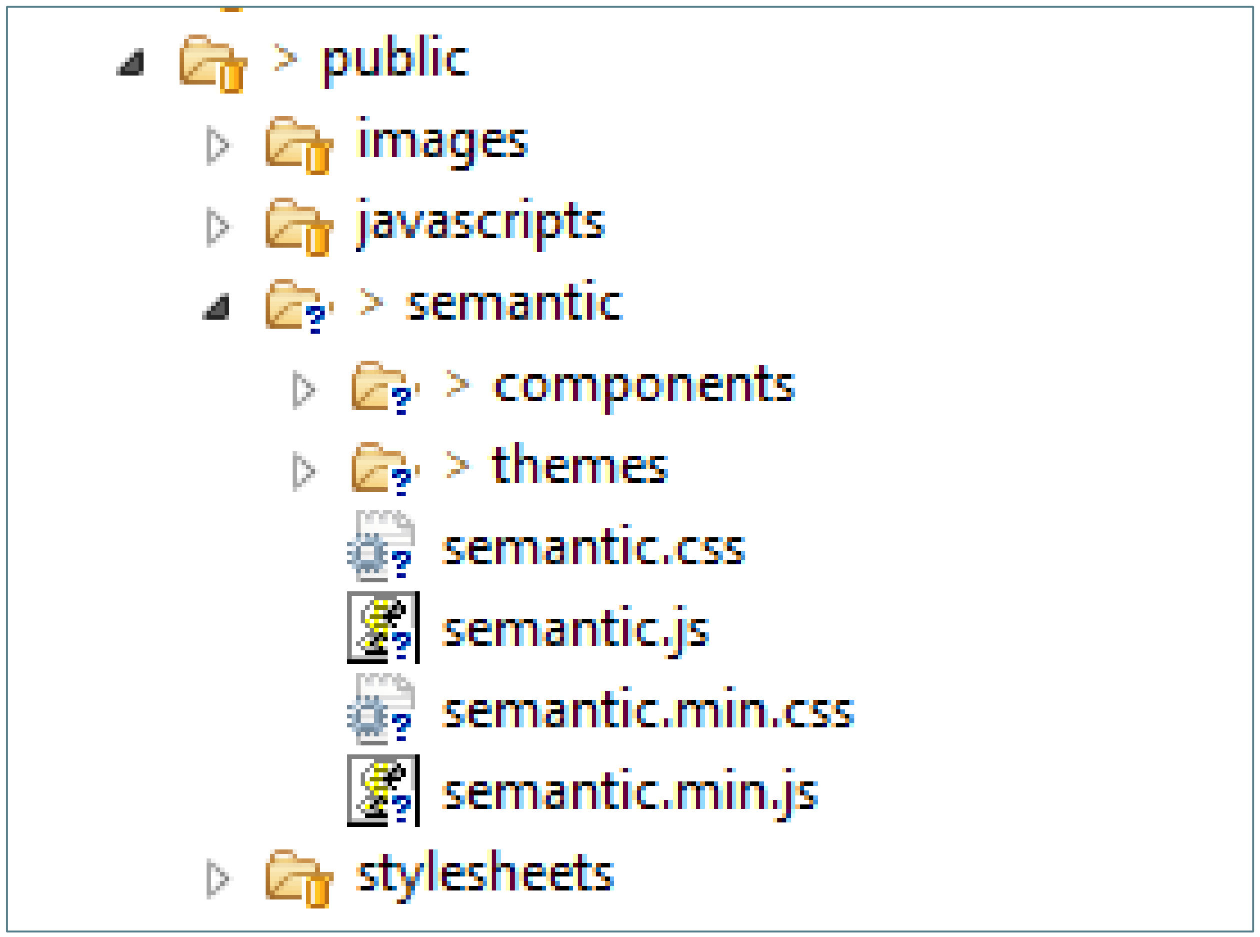
- [✓ Copy files and folders]
- Link to files and folders
- Link to files and recreate folder structure with virtual folders

✓ Create link locations relative to: PROJECT_LOC

- Once the folder is copied over rename it from **dist** to **semantic**. Your **public** folder structure should now look similar to this:

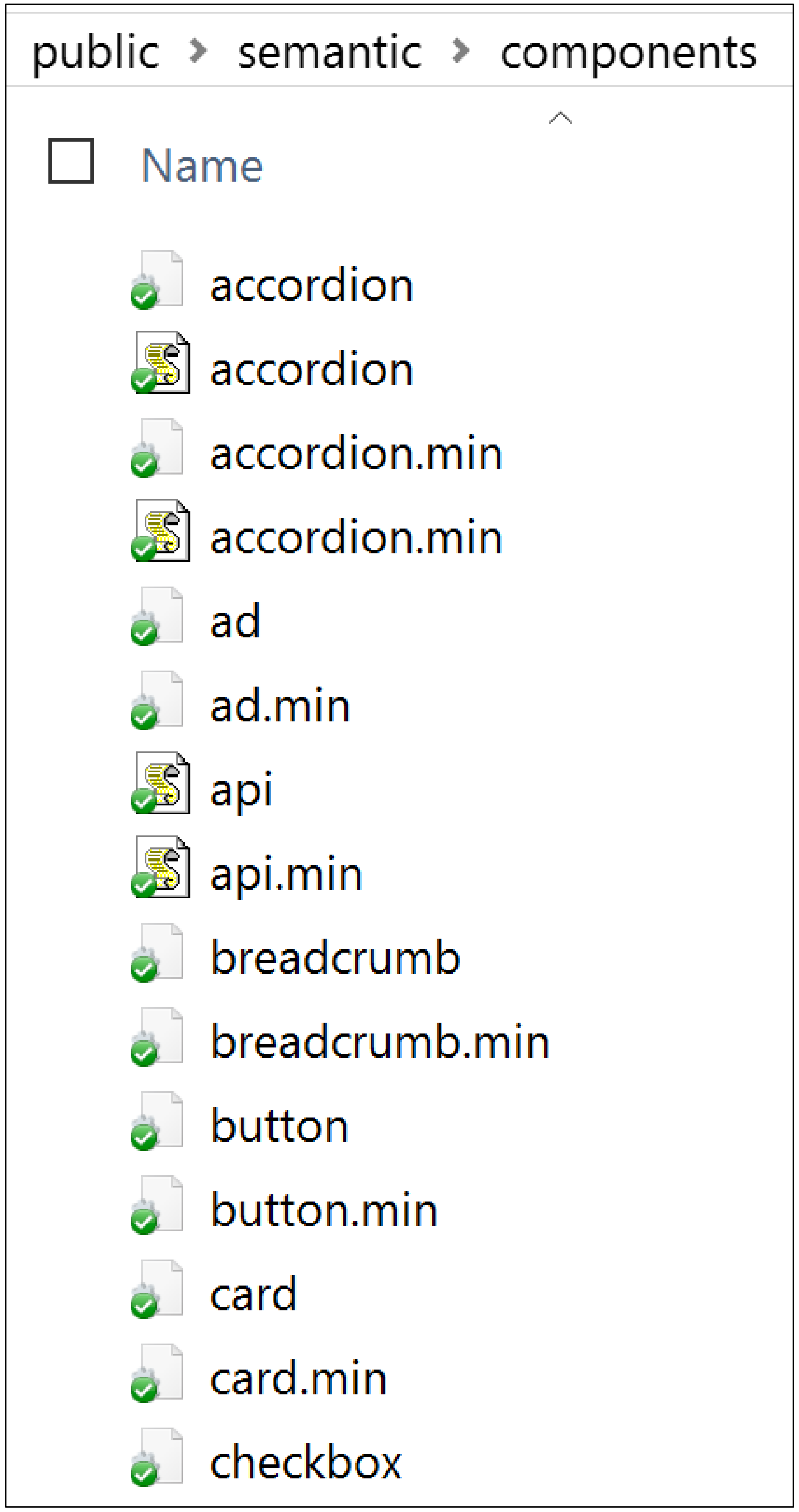


```
> public
    images
    javascripts
    > semantic
        > components
        > themes
        semantic.css
        semantic.js
        semantic.min.css
        semantic.min.js
    stylesheets
```



```
public
  images
  javascripts
  semantic
    components
    themes
    semantic.css
    semantic.js
    semantic.min.css
    semantic.min.js
  stylesheets
```

50+ UI elements
3000+ CSS variables



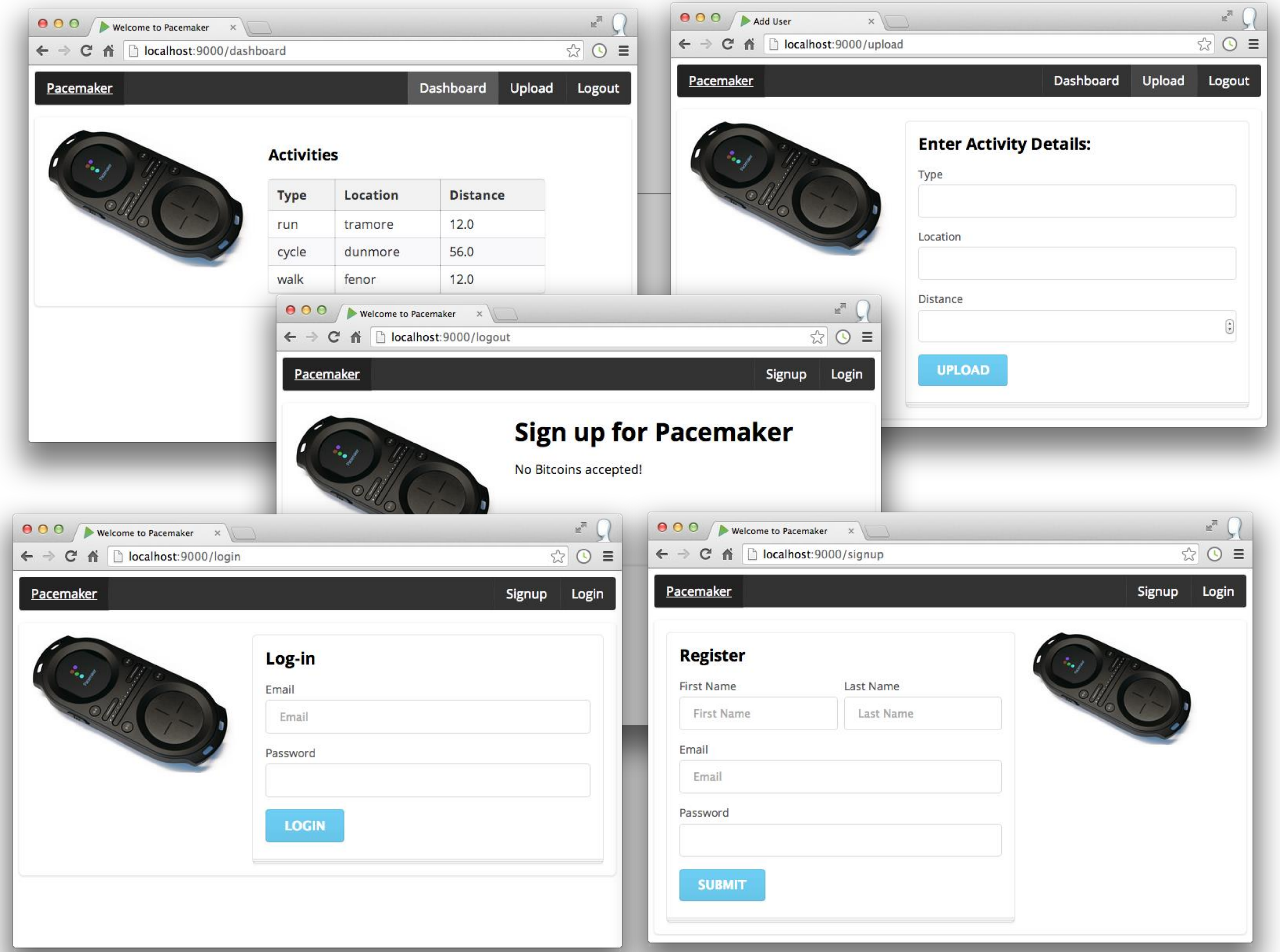
public › semantic › components

Name

```
accordion
accordion
accordion.min
accordion.min
ad
ad.min
api
api.min
breadcrumb
breadcrumb.min
button
button.min
card
card.min
checkbox
```

Topics List

- Overview of Semantic
- Specifics on Semantic
- Installing Semantic with Play
- Using Semantic with Play


Welcome to Pacemaker

localhost:9000/dashboard

Pacemaker Dashboard Upload Logout

Activities

Type	Location	Distance
run	tramore	12.0
cycle	dunmore	56.0
walk	fenor	12.0

Add User

localhost:9000/upload

Pacemaker Dashboard Upload Logout

Enter Activity Details:

Type

Location

Distance

UPLOAD

Welcome to Pacemaker

localhost:9000/logout

Pacemaker Signup Login

Sign up for Pacemaker

No Bitcoins accepted!

Welcome to Pacemaker

localhost:9000/login

Pacemaker Signup Login

Log-in

Email

Email

Password

LOGIN

Welcome to Pacemaker

localhost:9000/signup

Pacemaker Signup Login

Register

First Name Last Name

First Name Last Name

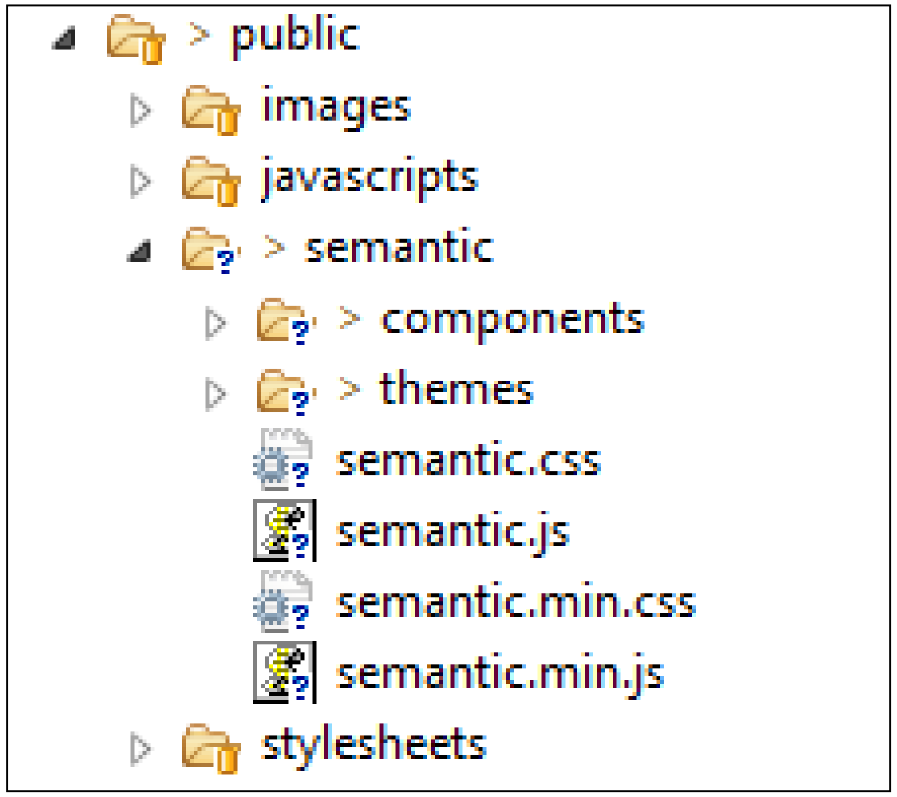
Email

Email

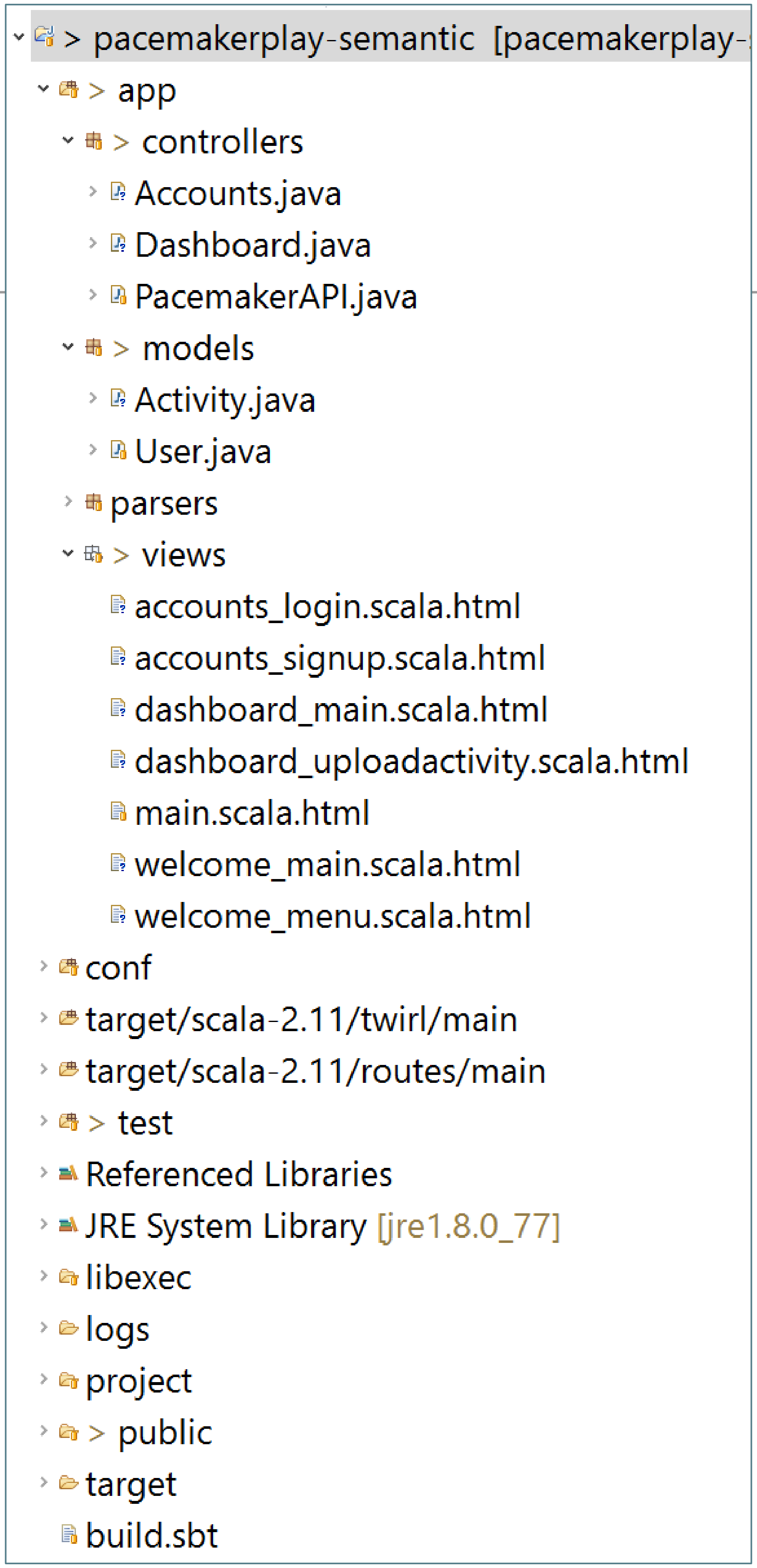
Password

SUBMIT

pacemakerplay-semantic

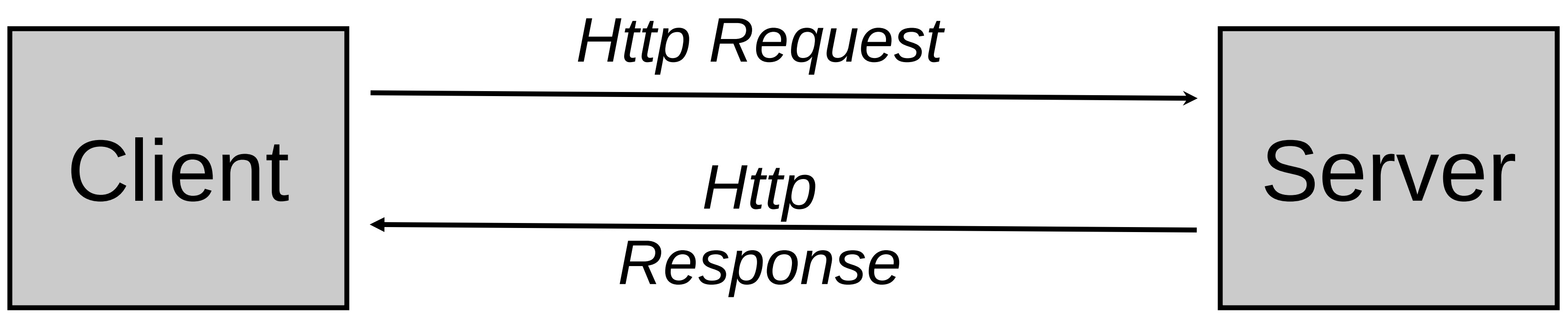


```
◢ > public
  ▷ images
  ▷ javascripts
  ◢ > semantic
    ▷ > components
    ▷ > themes
      semantic.css
      semantic.js
      semantic.min.css
      semantic.min.js
  ▷ stylesheets
```



```
⌄ > pacemakerplay-semantic  [pacemakerplay-
  ⌄ > app
    ⌄ > controllers
      › Accounts.java
      › Dashboard.java
      › PacemakerAPI.java
    ⌄ > models
      › Activity.java
      › User.java
    › parsers
    ⌄ > views
        accounts_login.scala.html
        accounts_signup.scala.html
        dashboard_main.scala.html
        dashboard_uploadactivity.scala.html
        main.scala.html
        welcome_main.scala.html
        welcome_menu.scala.html
  › conf
  › target/scala-2.11/twirl/main
  › target/scala-2.11/routes/main
  › > test
  › Referenced Libraries
  › JRE System Library [jre1.8.0_77]
  › libexec
  › logs
  › project
  › > public
  › target
    build.sbt
```

Web Applications - Request/Response

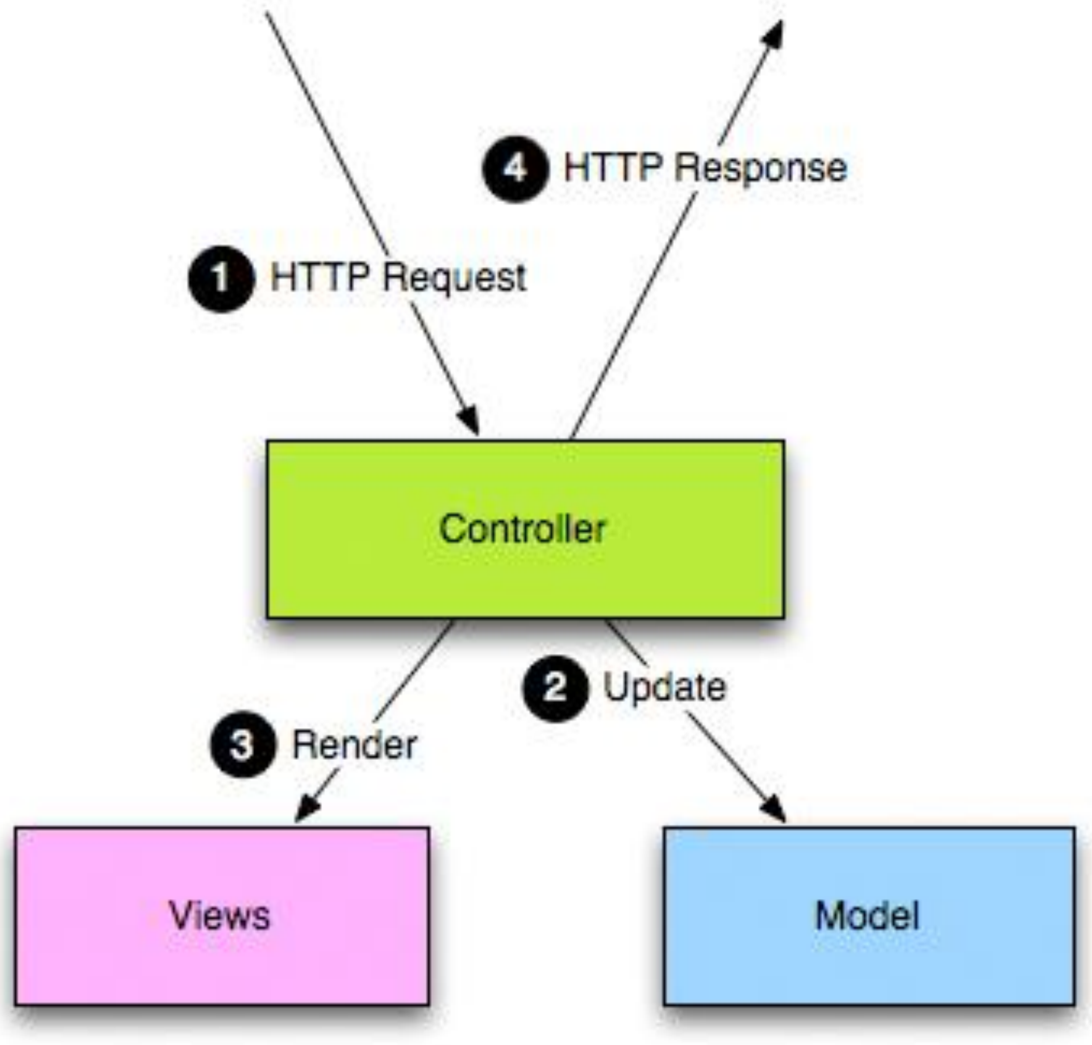


- Request - http request emitted by browser as a result to url in address bar, link, button or form submission on page.

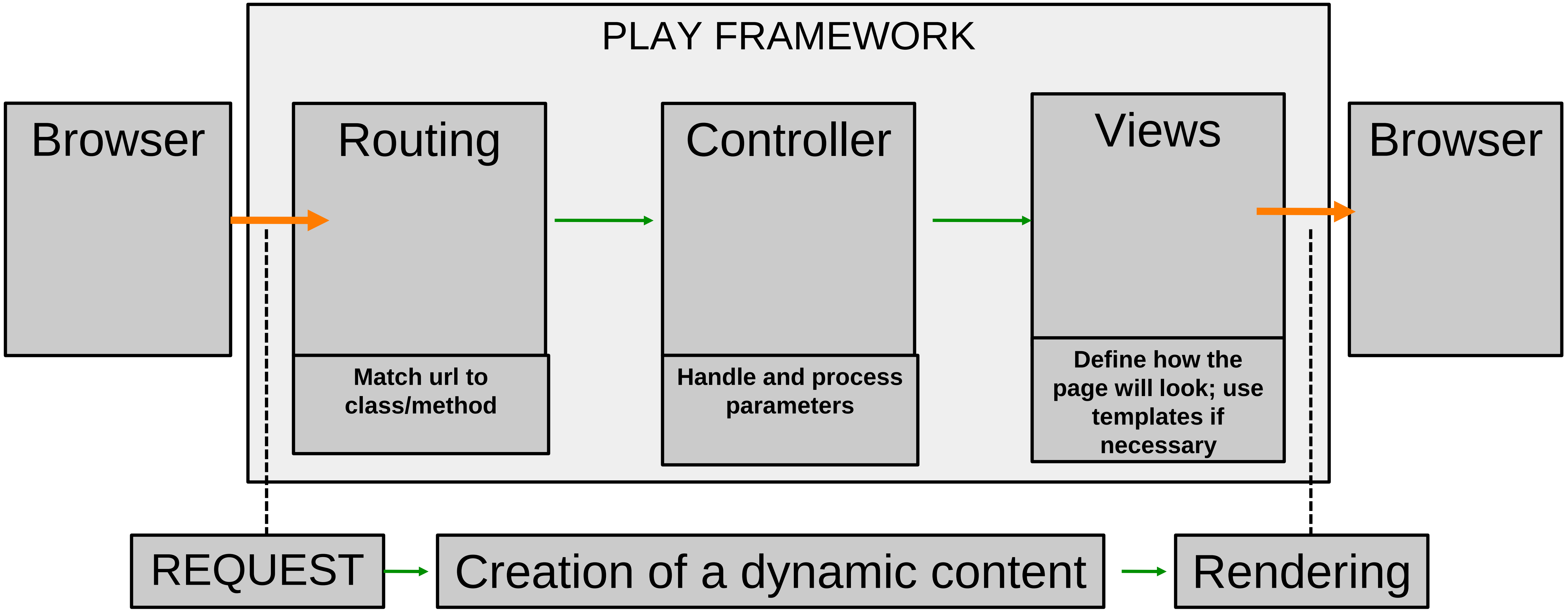
- Response - web page returned from service to be presented in browser.

Web Applications - MVC

- Model View Controller is a generally accepted pattern or separation of concerns within the server.
- **Model:** Core application domain model + database persistence.
- **View**: User Experience.
- **Controller**: Directly handle all requests, mediate with Model, build and assemble the response using the views.

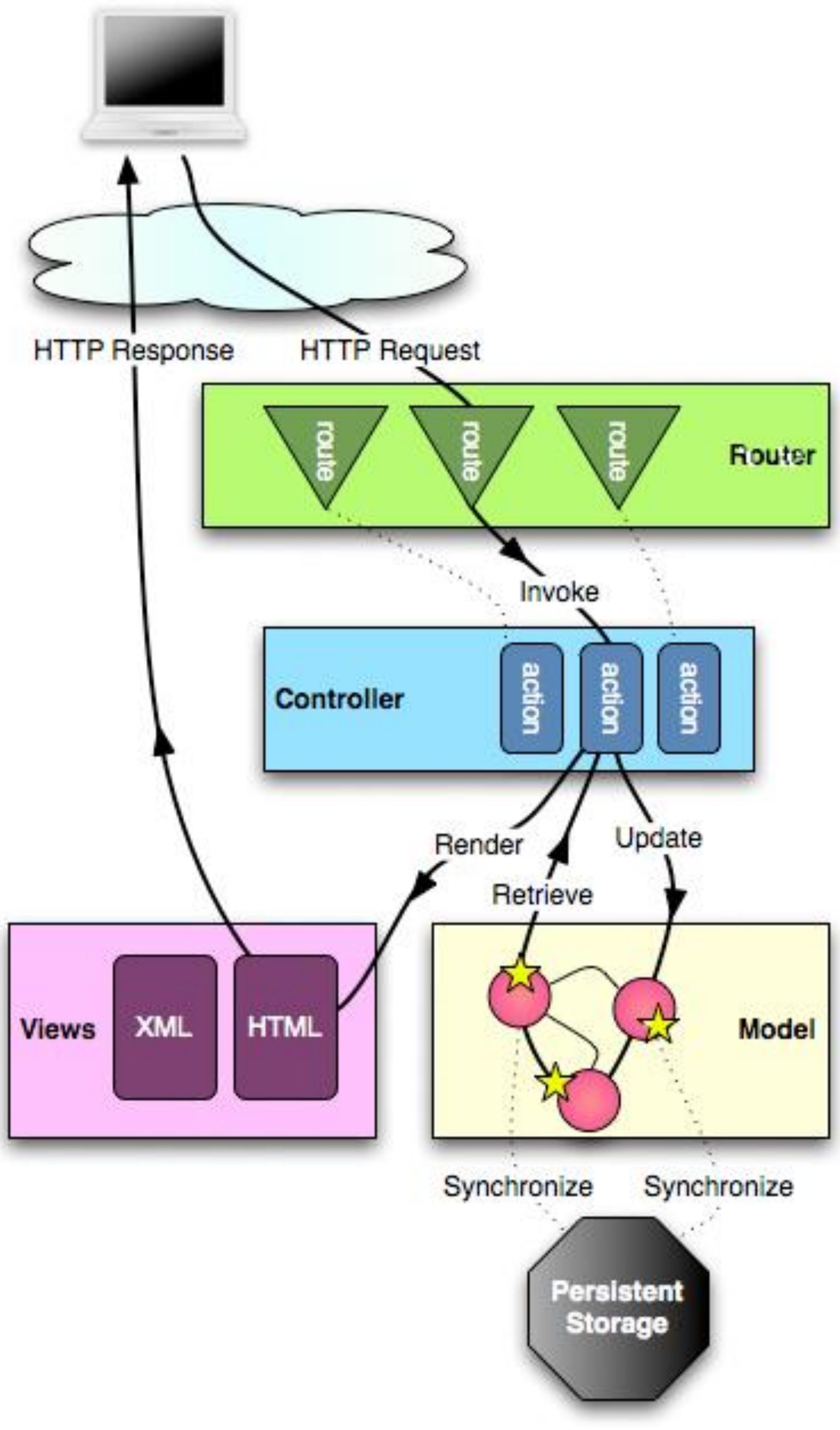


Request/Response Lifecycle


PLAY FRAMEWORK
Browser
Routing
Match url to class/method
Controller
Handle and process parameters
Views
Define how the page will look; use templates if necessary
Browser
REQUEST
Creation of a dynamic content
Rendering

MVC in Play

- **Router:** examine incoming requests and match to corresponding Controller/Action.
- **Action:** a method in the controller.

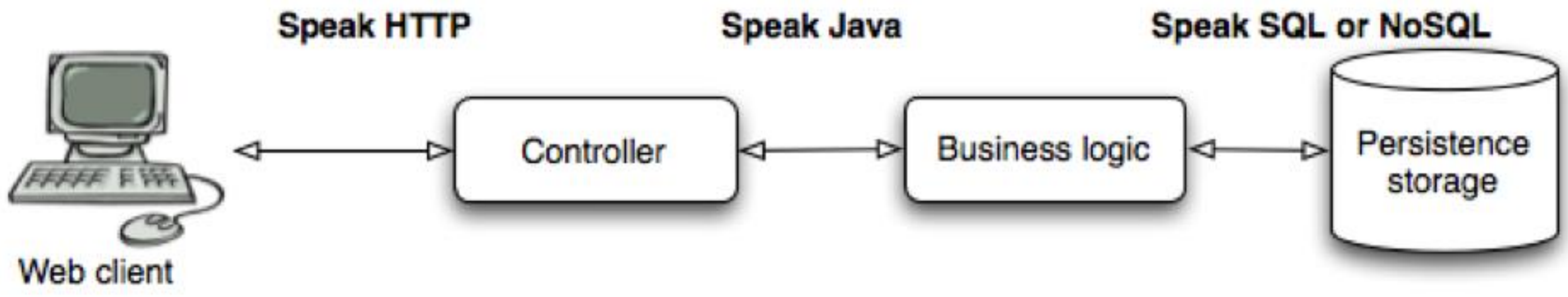


Routes - UI

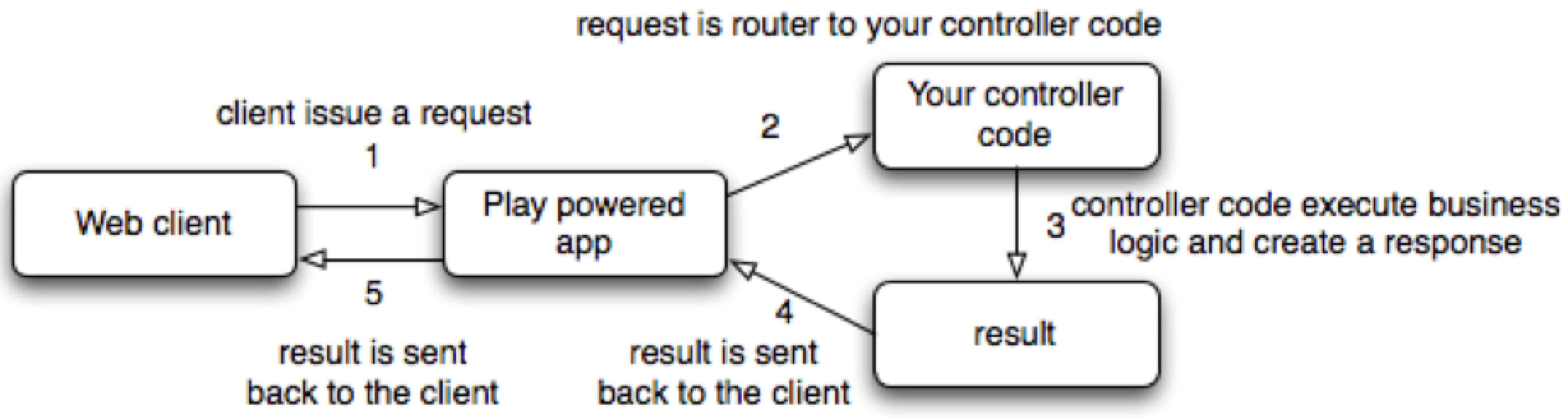
```
# UI

GET     /                           controllers.Accounts.index()
GET     /signup                     controllers.Accounts.signup()
GET     /login                      controllers.Accounts.login()
GET     /logout                     controllers.Accounts.logout()
POST    /register                   controllers.Accounts.register()
POST    /authenticate               controllers.Accounts.authenticate()

GET     /dashboard                  controllers.Dashboard.index()
GET     /upload                     controllers.Dashboard.uploadActivityForm()
POST    /submitactivity             controllers.Dashboard.submitActivity()
```

- Routes to deliver UI.
- Each of these routes appears in views.
- Each of these actions generates and returns a complete HTML page.

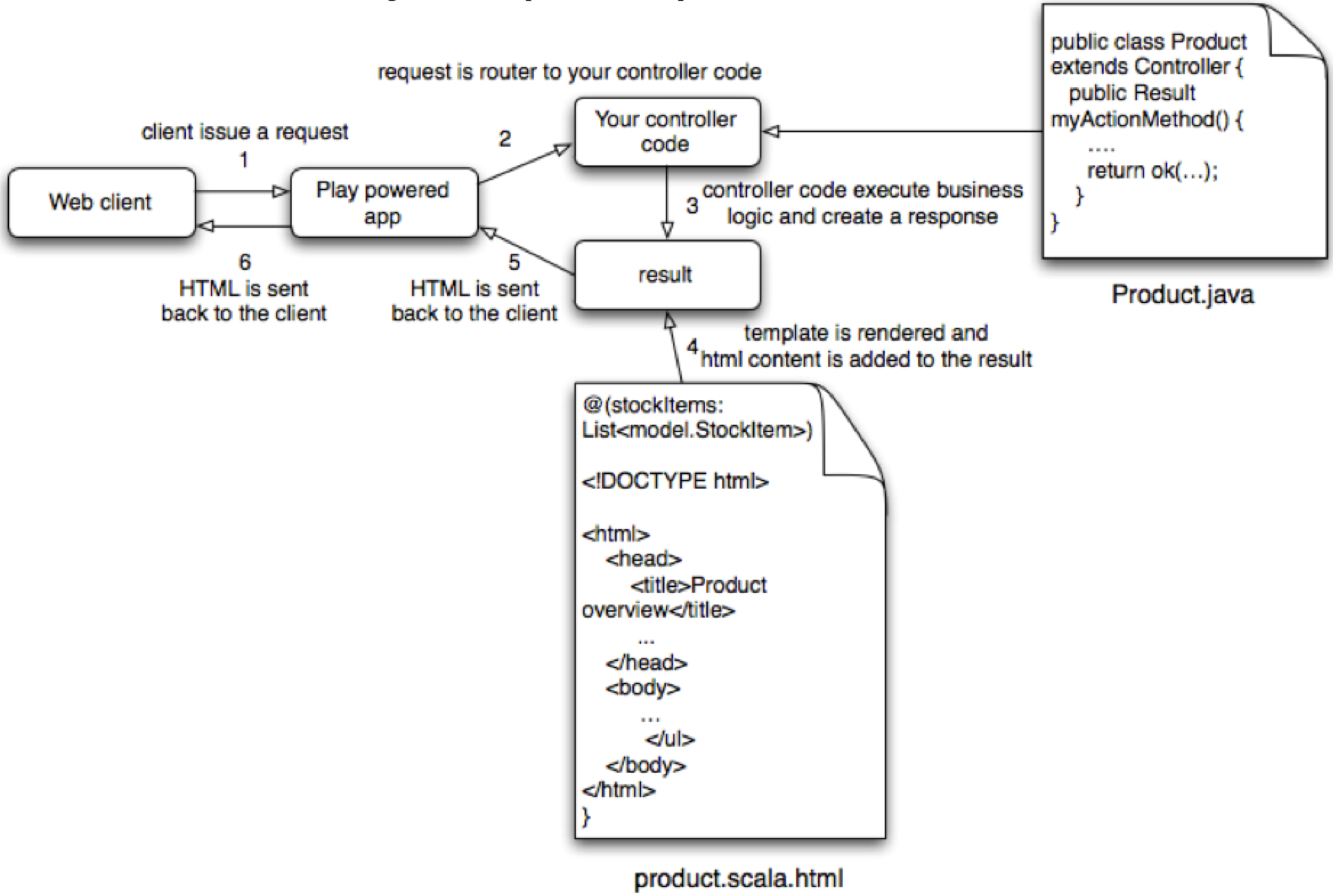
Role of Controller



Controller Lifecycle


request is router to your controller code
client issue a request
1
2
Your controller code
Web client
Play powered app
3
controller code execute business logic and create a response
5
4
result
result is sent back to the client
result is sent back to the client

Controller Lifecycle (detail)

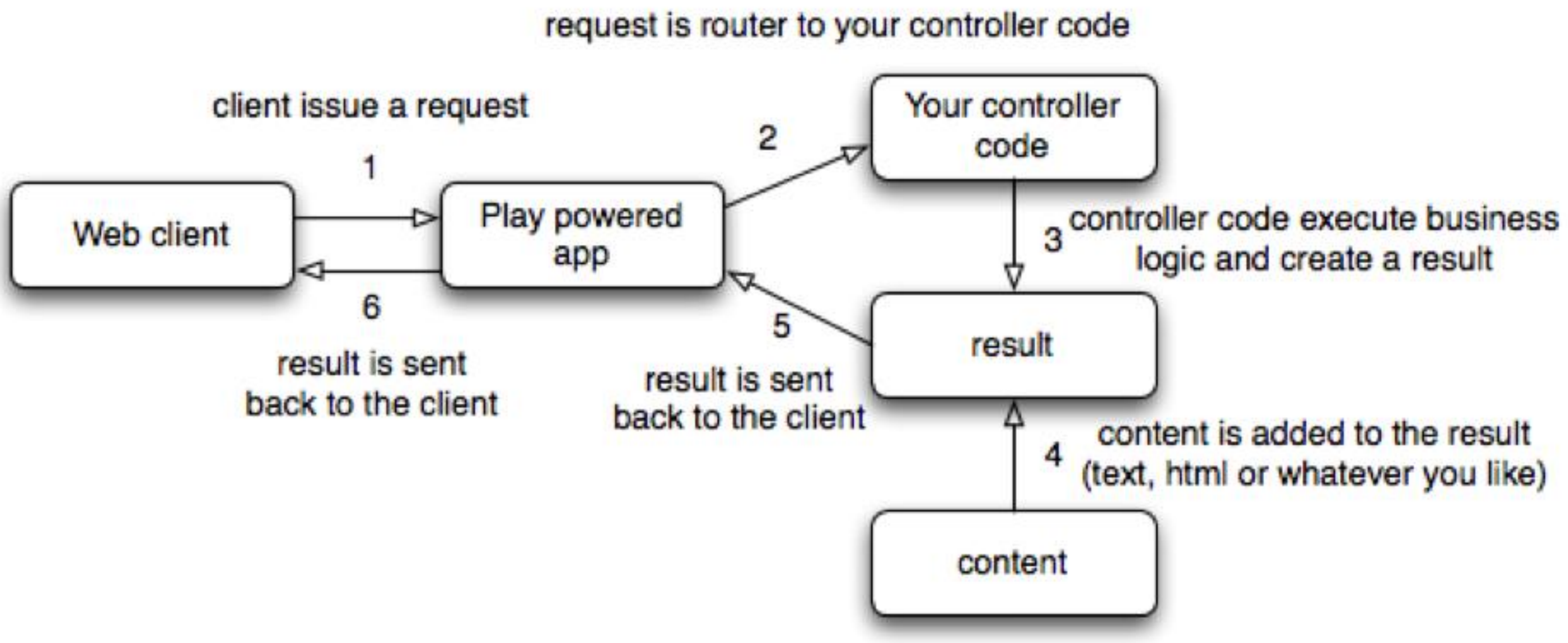


request is router to your controller code
client issue a request
1
2
Web client
Play powered app
Your controller code
3
controller code execute business logic and create a response
result
4
template is rendered and html content is added to the result
5
HTML is sent back to the client
6
HTML is sent back to the client
Product.java
product.scala.html

```
public class Product
extends Controller {
   public Result
myActionMethod() {
      ....
      return ok(...);
   }
}
```

```
@(stockItems:
List<model.StockItem>)

<!DOCTYPE html>

<html>
   <head>
      <title>Product
overview</title>
        ...
   </head>
   <body>
        ...
        </ul>
   </body>
</html>
}
```

Controller Lifecycle with Content


request is router to your controller code
client issue a request
1
Web client
Play powered app
2
Your controller code
3
controller code execute business logic and create a result
result
5
6
result is sent back to the client
result is sent back to the client
4
content is added to the result (text, html or whatever you like)
content

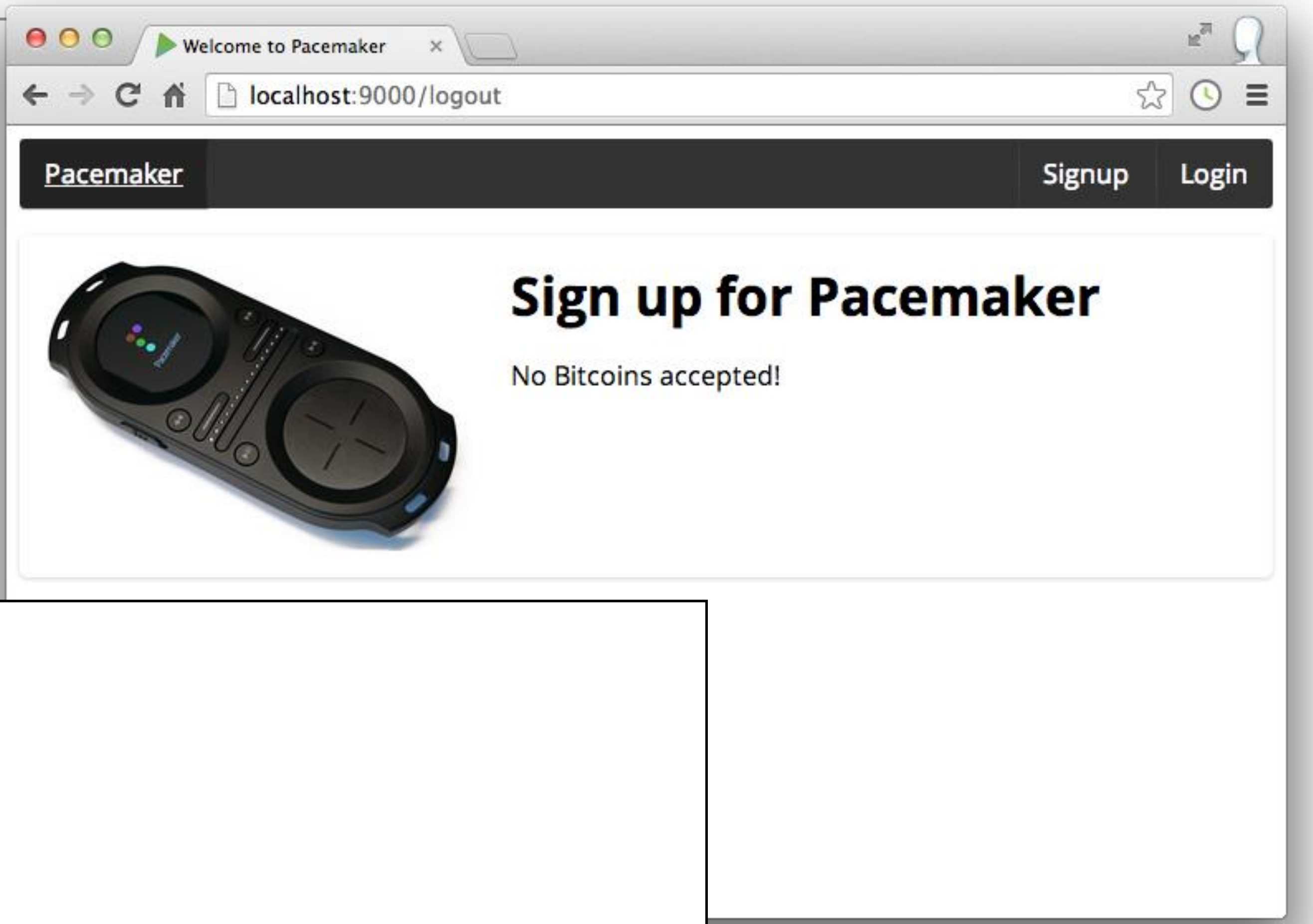
Welcome

routes

```
GET     /                 controllers.Accounts.index()
```

Accounts.java

```
public class Accounts extends Controller
{
  public Result index()
  {
    return ok(welcome_main.render());
  }
...
}
```


Welcome to Pacemaker

localhost:9000/logout

Sign up for Pacemaker

No Bitcoins accepted!

```
@()

@main("Welcome to Pacemaker") {
  @welcome_menu()

<section class="ui raised segment">
  <div class="ui grid">
    <aside class="six wide column">
      <img src="@routes.Assets.at("images/pacemaker.jpg")" class="ui medium image">
    </aside>
    <article class="ten wide column">
      <h1 class="ui  header"> Sign up for Pacemaker  </h1>
      <p> No Bitcoins accepted! </p>
    </article>
  </div>
</section>
}
```

welcome_main.scala.html

templating

Entire view is inserted into **@content** section of page

```
@(title: String)(content: Html)
<!DOCTYPE html>
<html>
        <head>
          <title>@title</title>
          <meta charset="utf-8">
          <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
          <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.

          <link rel="stylesheet" type="text/css" href="@routes.Assets.at("semantic/css/semantic.css"
          <link rel="stylesheet" type="text/css" href="@routes.Assets.at("stylesheets/main.css")">
          <link href='http://fonts.googleapis.com/css?family=Source+Sans+Pro:400,700|Open+Sans:
    <link rel="shortcut icon" type="image/png" href="@routes.Assets.at("images/favicon.png")">

    <script src="@routes.Assets.at("javascripts/jquery-2.0.3.min.js")"></script>
    <script src="@routes.Assets.at("semantic/javascript/semantic.min.js")"></script>
        </head>

  <body>
    @content
   </body>
</html>
```

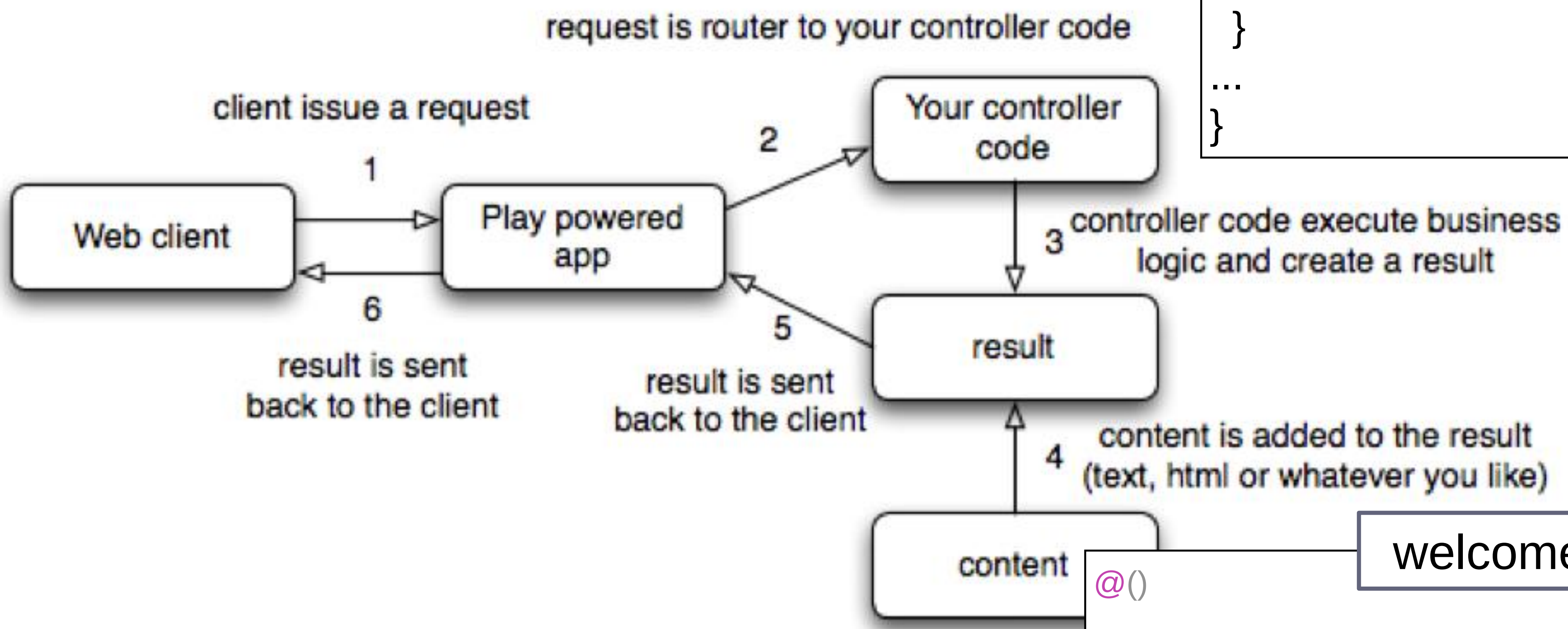
main.scala.html

```
@()

@main("Welcome to Pacemaker") {
  @welcome_menu()

<section class="ui raised segment">
  <div class="ui grid">
    <aside class="six wide column">
      <img src="@routes.Assets.at("images/pacemaker.jpg")" class="ui medium image">
    </aside>
    <article class="ten wide column">
      <h1 class="ui  header"> Sign up for Pacemaker  </h1>
      <p> No Bitcoins accepted! </p>
    </article>
  </div>
</section>
}
```

welcome_main.scala.html

@main implies that main.scala.html will define the structure of the generated page

```
GET  /    controllers.Accounts.index()
```


request is router to your controller code
client issue a request
1
2
Web client
Play powered app
Your controller code
3
controller code execute business logic and create a result
result
6
5
result is sent back to the client
result is sent back to the client
4
content is added to the result (text, html or whatever you like)
content

Accounts.java

```
public class Accounts extends Controller
{
  public Result index()
  {
    return ok(welcome_main.render());
  }
...
}
```

welcome_main.scala.html

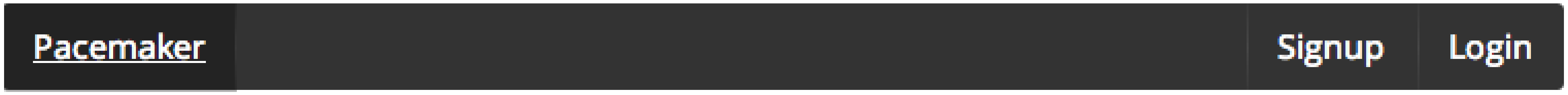
```
@()

@main("Welcome to Pacemaker") {
  @welcome_menu()

<section class="ui raised segment">
  <div class="ui grid">
    <aside class="six wide column">
      <img src="@routes.Assets.at("images/pacemaker.jpg")" cl
    </aside>
    <article class="ten wide column">
      <h1 class="ui  header"> Sign up for Pacemaker  </h1>
      <p> No Bitcoins accepted! </p>
    </article>
  </div>
</section>
}
```

Includes

```
@()

@main("Welcome to Pacemaker") {
  @welcome_menu()

<section class="ui raised segment">
ges/pacemaker.jpg")" class="ui medium image">
 Pacemaker  </h1>
  </div>
</section>
}
```

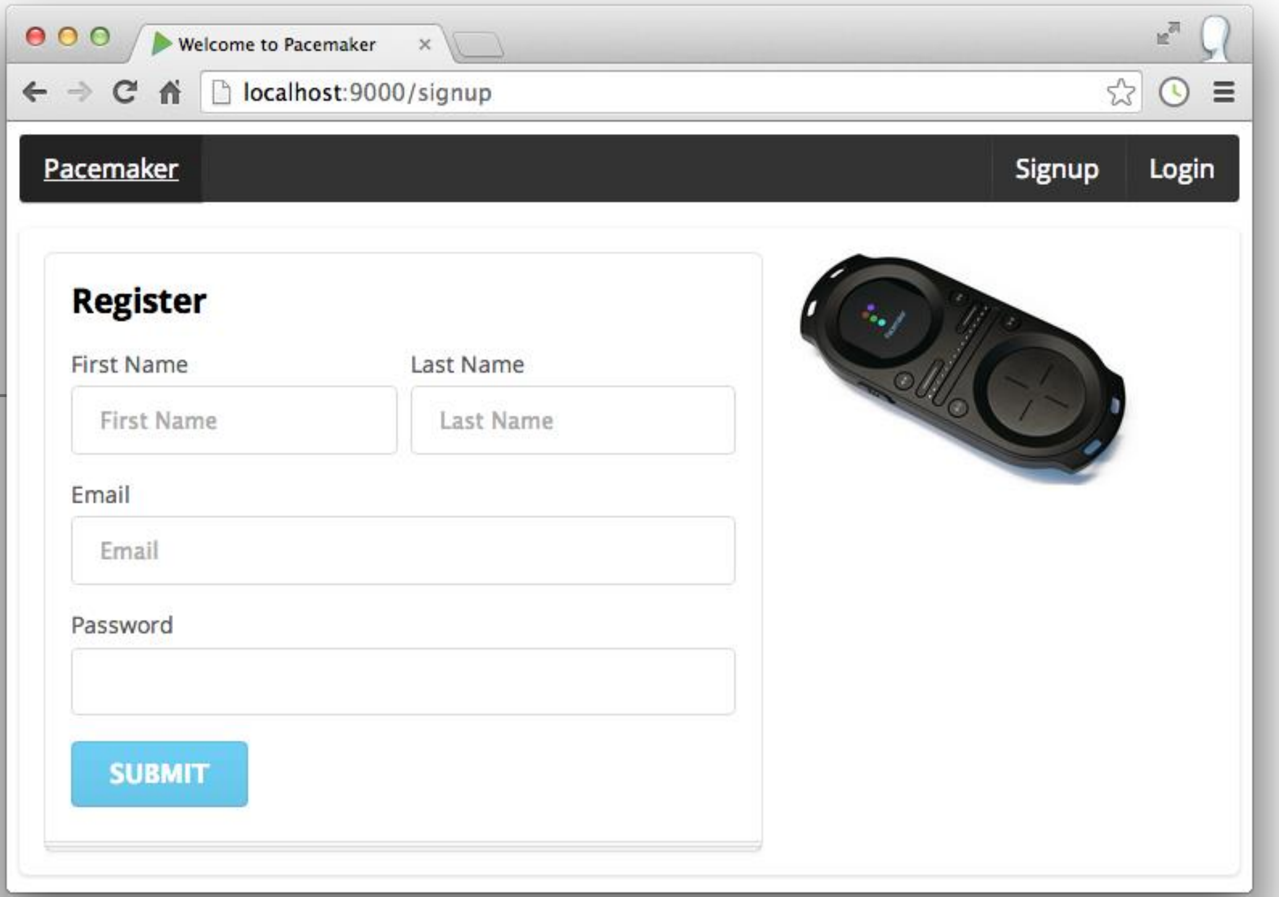
```
@()

<nav class="ui inverted menu">
  <div class="header item"> <a href="/"> Pacemaker </a> </div>
  <div class="right menu">
    <a class="item" href="/signup"> Signup</a>
    <a class="item" href="/login">  Login</a>
  </div>
</nav
```

welcome_menu.scala.html



Signup

```
@()

@main("Welcome to Pacemaker") {
  @welcome_menu()

<section class="ui raised segment">
  <div class="ui grid">
    <div class="ui ten wide column">
      <div class="ui stacked fluid form segment">
        <form action="/register" method="POST">
          <h3 class="ui header">Register</h3>
          <div class="two fields">
            <div class="field">
              <label>First Name</label>
              <input placeholder="First Name" type="text" name="firstname">
            </div>
            <div class="field">
              <label>Last Name</label>
              <input placeholder="Last Name" type="text" name="lastname">
            </div>
          </div>
          <div class="field">
            <label>Email</label>
            <input placeholder="Email" type="text" name="email">
          </div>
          <div class="field">
            <label>Password</label>
            <input type="password" name="password">
          </div>
          <button class="ui blue submit button">Submit</button>
        </form>
      </div>
    </div>
    <aside class="ui five wide column">
      <img src="@routes.Assets.at("images/pacemaker.jpg")" class="ui medium image">
    </aside>
  </div>
</section>
}
```



```
GET     /signup       controllers.Accounts.signup()
```

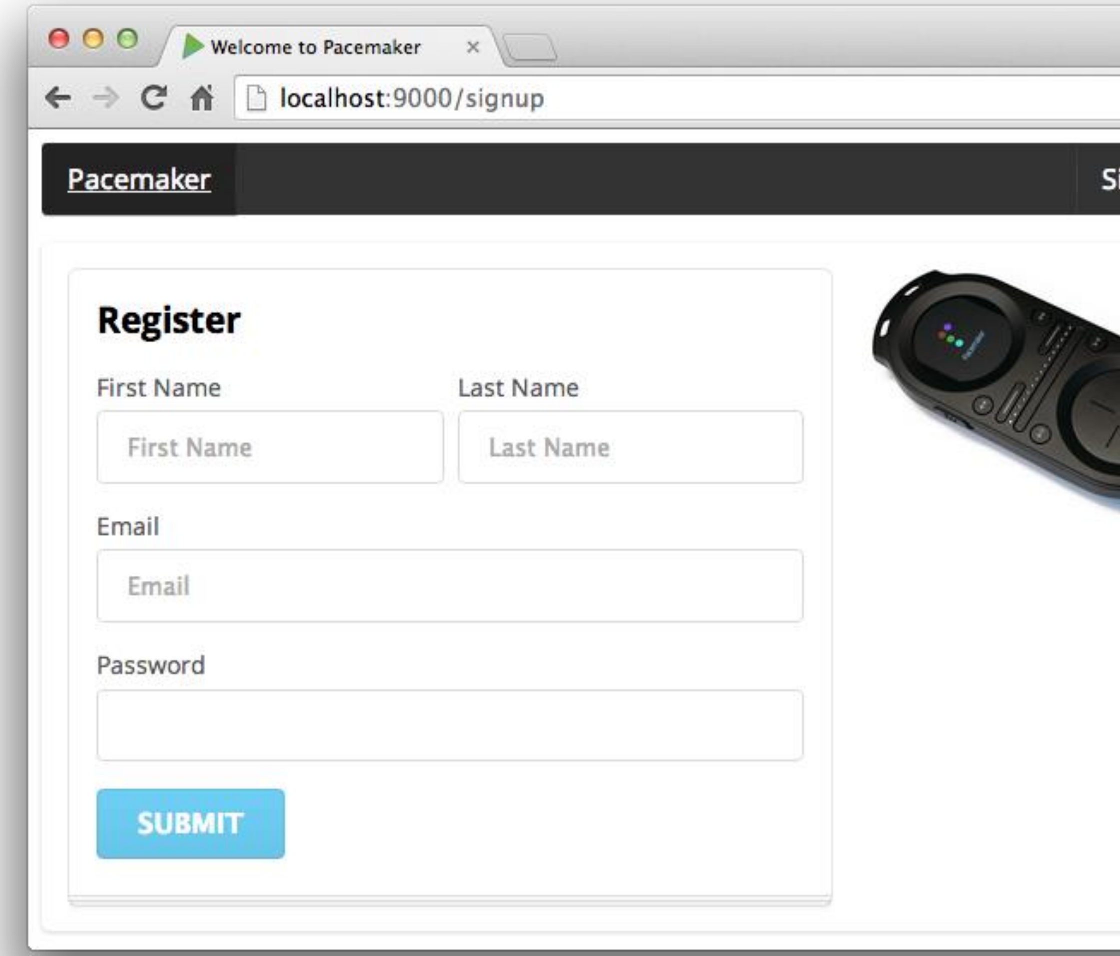
```
public Result signup()
{
  return ok(accounts_signup.render());
}
```

Signup

```
GET     /signup        controllers.Accounts.signup()
```

```
<form action="/register" method="POST">
  <h3 class="ui header">Register</h3>
  <div class="field">
  <div class="two fields">
    <div class="field">
      <label>First Name</label>
      <input placeholder="First Name" type="text"   name="firstname" >
    </div>
    <div class="field">
      <label>Last Name</label>
      <input placeholder="Last Name" type="text"   name="lastname">
    </div>
  </div>
  <div class="field">
    <label>Email</label>
    <input placeholder="Email"     type="text"     name="email">
  </div>
  <div class="field">
    <label>Password</label>
    <input type="password"                          name="password" >
  </div>
  <button class="ui blue submit button">Submit</button>
</form>
```

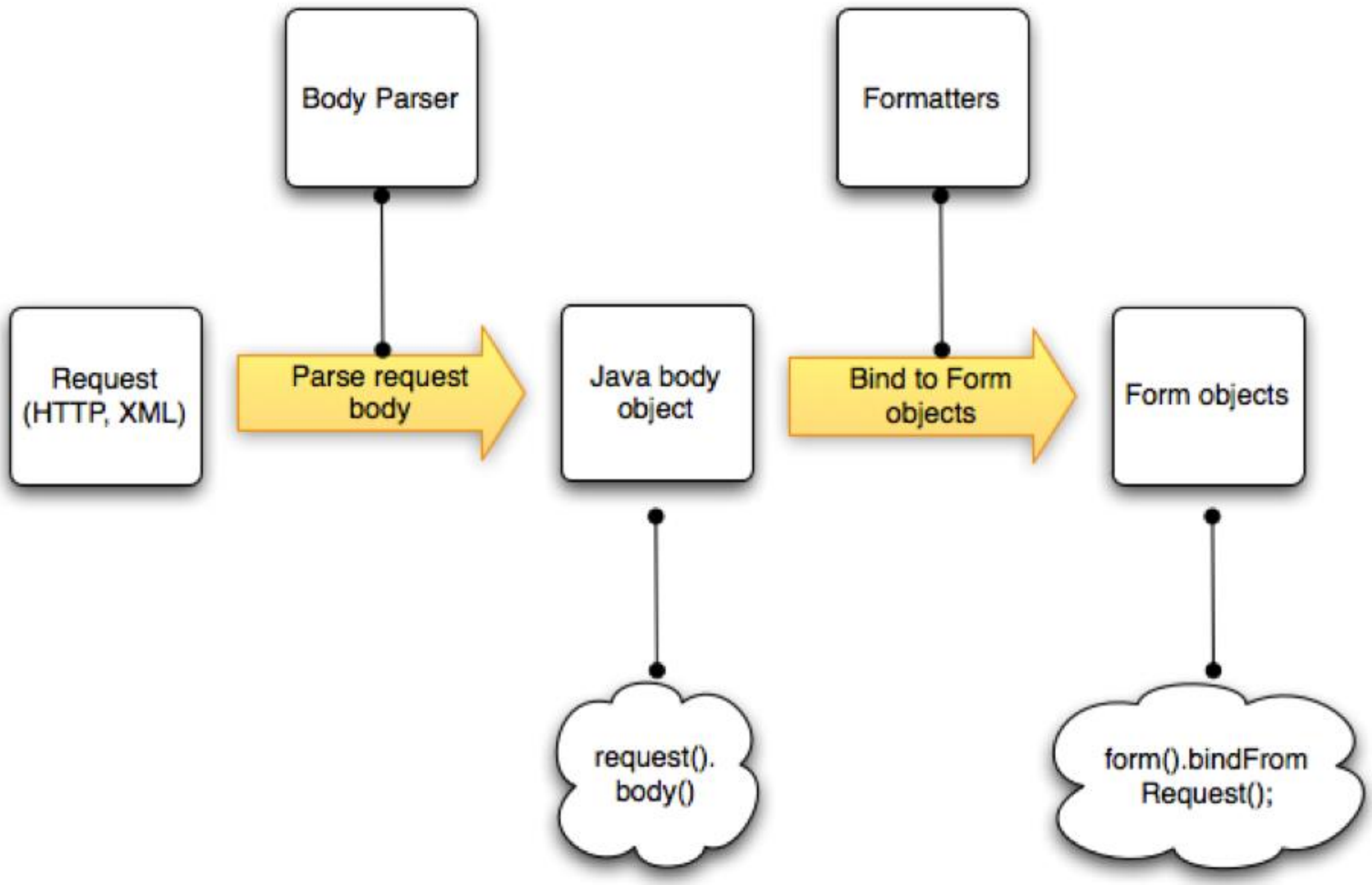
accounts_signup.scala.html



```
public Result signup()
{
  return ok(accounts_signup.render());
}
```

Signup

```
public class Accounts extends Controller
{
   private static Form<User> userForm;
   private static Form<User> loginForm;

 //…
  public Result register()
  {
       Form<User> boundForm = userForm.bindFromRequest();
       if(loginForm.hasErrors())
    {
      return badRequest(accounts_login.render());
    }
    else
    {
      User user = boundForm.get();
      Logger.info ("User = " + user.toString());
      user.save();
      return ok(welcome_main.render());
    }
  }

//...
```


Body Parser
Formatters
Request (HTTP, XML)
Parse request body
Java body object
Bind to Form objects
Form objects
request(). body()
form().bindFrom Request();

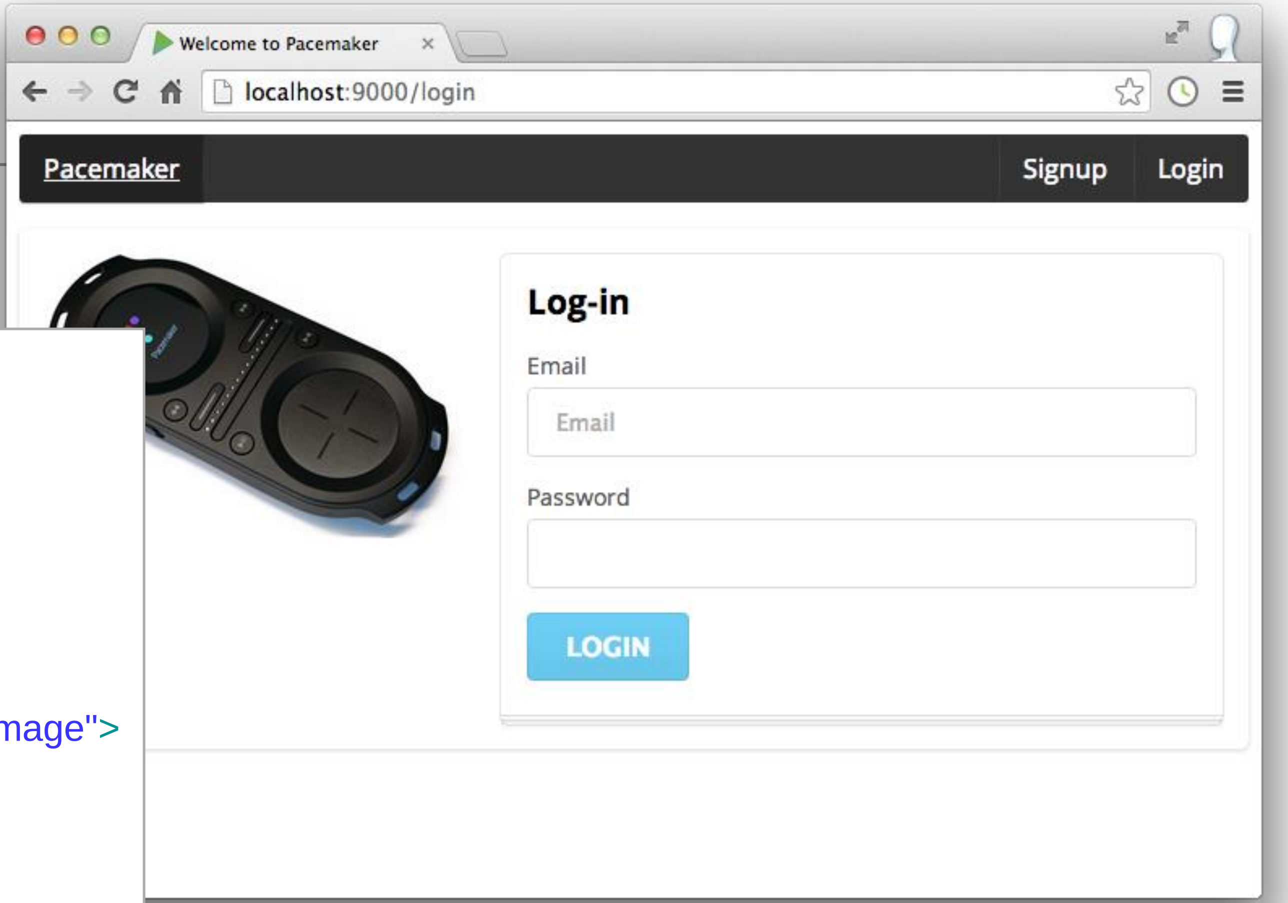
Signup Form Processing

- Recover named form input items from request.
- Extract these elements into a Java object

```
public class Accounts extends Controller
{
   private static Form<User> userForm;
   private static Form<User> loginForm;

 //…
  public Result register()
  {
      Form<User> boundForm = userForm.bindFromRequest();
      if(loginForm.hasErrors())
    {
      return badRequest(accounts_login.render());
    }
    else
    {
      User user = boundForm.get();
      Logger.info ("User = " + user.toString());
      user.save();
      return ok(welcome_main.render());
    }
  }

//...
```

```
GET     /login          controllers.Accounts.login()
```

Login


Welcome to Pacemaker

localhost:9000/login

```
@()

@main("Welcome to Pacemaker") {
  @welcome_menu()

 <section class="ui raised segment">
   <div class="ui grid">
     <aside class="ui six wide column">
      <img src="@routes.Assets.at("images/pacemaker.jpg")" class="ui medium image">
     </aside>
     <div class="ui ten wide column fluid form">
       <div class="ui stacked segment">
         <form action="/authenticate" method="POST">
           <h3 class="ui header">Log-in</h3>
           <div class="field">
             <label>Email</label>
             <input placeholder="Email" type="text" name="email">
           </div>
           <div class="field">
             <label>Password</label>
             <input type="password" name="password">
           </div>
           <button class="ui blue submit button">Login</button>
         </form>
       </div>
     </div>
   </div>
</section>
}
```

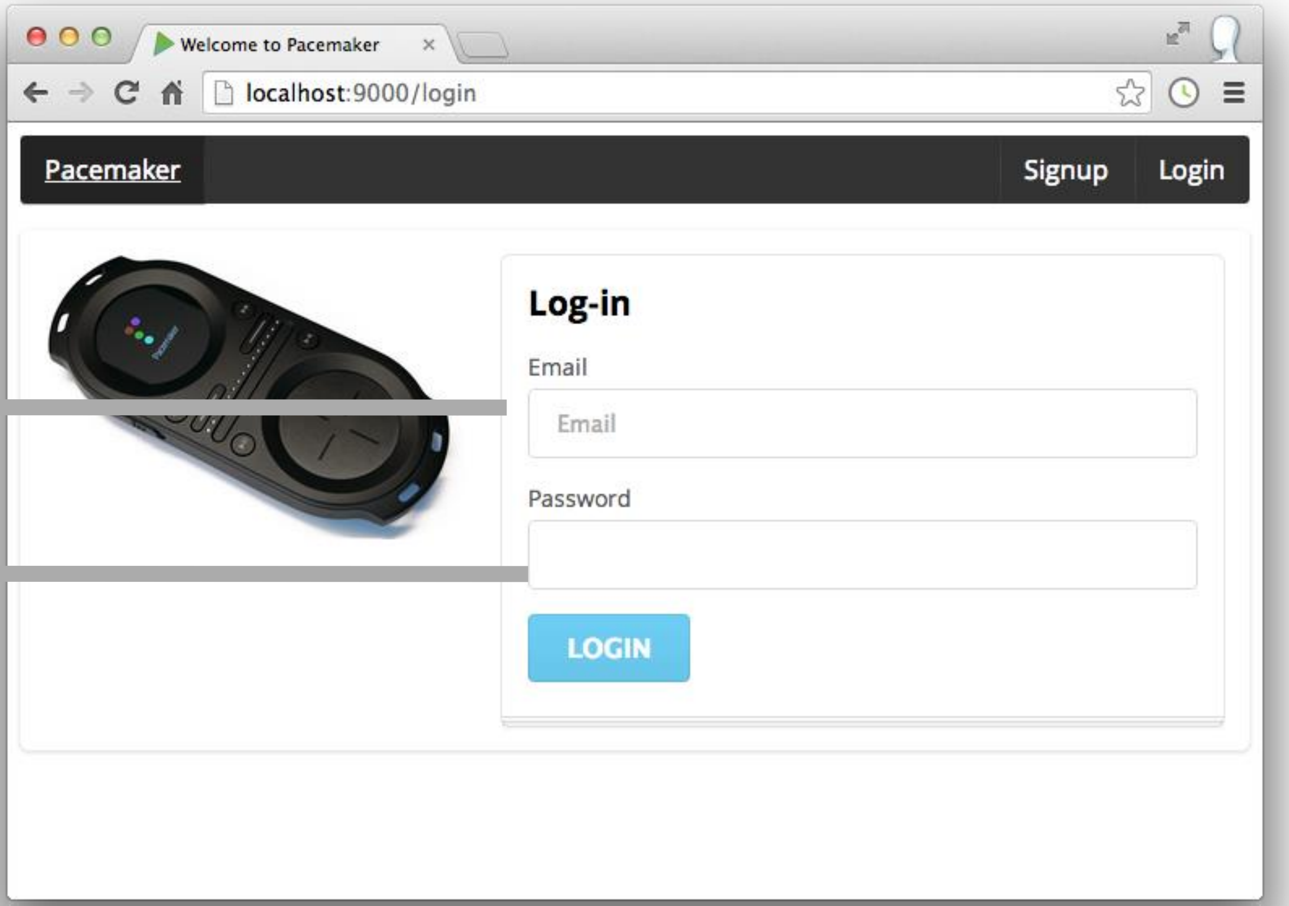
accounts_login.scala.html

```
public Result login()
{
  return ok(accounts_login.render());
}
```

Login

```
<form action="/authenticate" method="POST">
  <h3 class="ui header">Log-in</h3>
  <div class="field">
    <label>Email</label>
    <input placeholder="Email" type="text" name="email">
  </div>
  <div class="field">
    <label>Password</label>
    <input type="password"                  name="password">
  </div>
  <button class="ui blue submit button">Login</button>
</form>
```

accounts_login.scala.html



Sessions - login

- A globally accessible data structure into which we put details of ‘current’ user.
- Read this back in other controllers to determine appropriate content.

```
public class Accounts extends Controller
{
  private static Form<User> loginForm;
  //...

  public Result authenticate()
  {
    Form<User> boundForm = loginForm.bindFromRequest();

    if(loginForm.hasErrors())
    {
      return badRequest(accounts_login.render());
    }
    else
    {
      session("email", boundForm.get().email);
      return redirect(routes.Dashboard.index());
    }
  }
  //...
}
```

- Not checking if user is valid!
- Should compare password/email with database, and only allow in of valid user credential presented.

Sessions - Logout

- Destroy the session.
- Redirect to Welcome page.

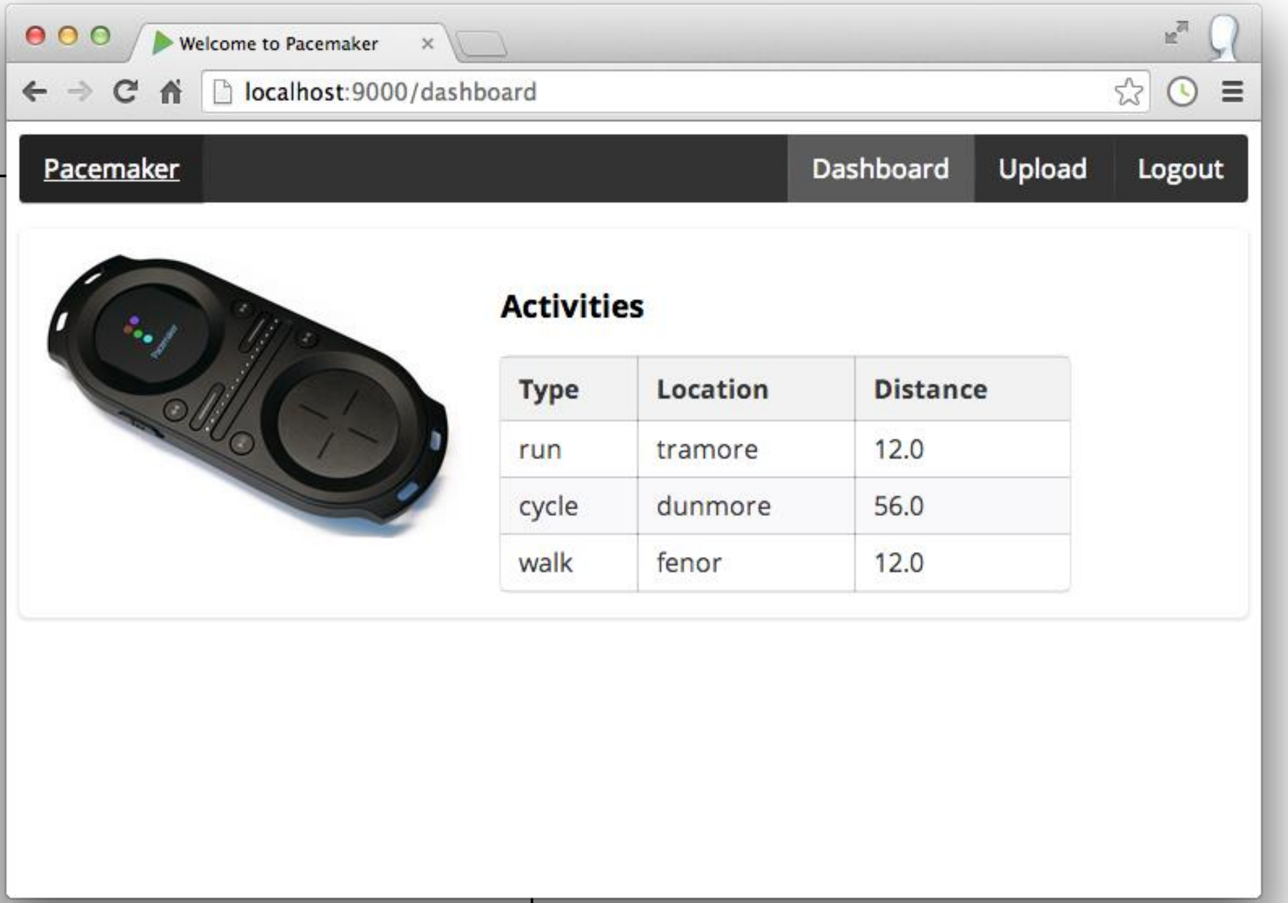
```
public Result logout()
{
  session().clear();
  return ok(welcome_main.render());
}
```

Dashboard

```
@(activities: List[Activity])

@main("Welcome to Pacemaker") {

  <nav class="ui inverted menu">
    <div class="header item"> <a href="/"> Pacemaker </a> </div>
    <div class="right menu">
      <a class="active item" href="/dashboard"> Dashboard</a>
      <a class="item" href="/upload">  Upload</a>
      <a class="item" href="/logout">  Logout</a>
    </div>
  </nav>

  <section class="ui raised segment">
    <div class="ui grid">
      <aside class="six wide column">
        <img src="@routes.Assets.at("images/pacemaker.jpg")" class="ui medium image">
      </aside>
      <article class="eight wide column">
        <h3> <class="ui header"> Activities </h3>
        <table class="ui celled table segment">
          <thead>
            <tr>
              <th>Type</th>
              <th>Location</th>
              <th>Distance</th>
            </tr>
          </thead>
          <tbody>
                    @for(i <- 0 until activities.size) {
                      <tr>
                        <td> @activities(i).kind </td> <td> @activities(i).location </td> <td> @activities(i).distance </td>
                      </tr>
                    }
          </tbody>
        </table>
      </article>
    </div>
  </section>
}
```

dashboard_main.scala.html

GET /dashboard controllers.Dashboard.index()



Activities

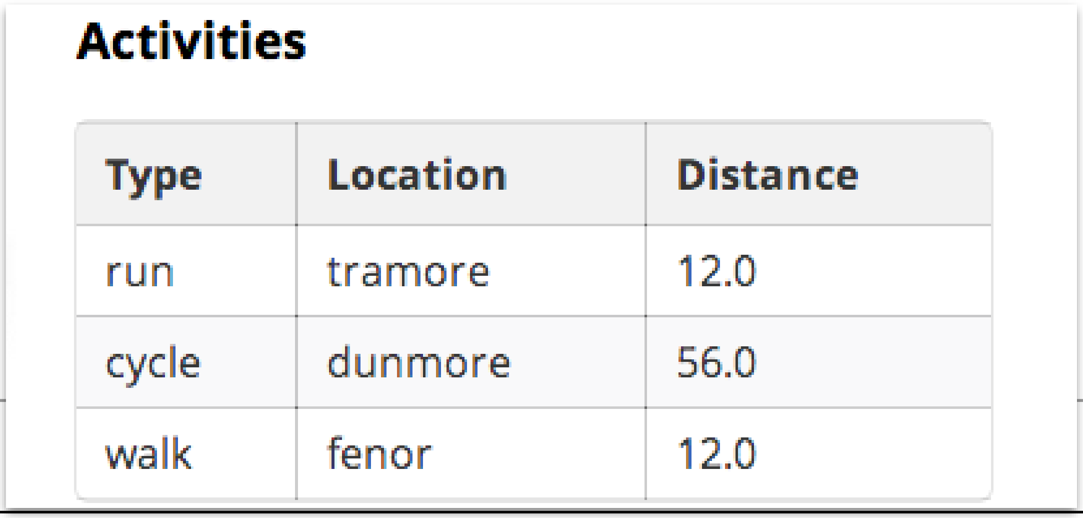
Type	Location	Distance
run	tramore	12.0
cycle	dunmore	56.0
walk	fenor	12.0

Dashboard

```
GET     /dashboard              controllers.Dashboard.index()
GET     /upload                 controllers.Dashboard.uploadActivityForm()
POST    /submitactivity         controllers.Dashboard.submitActivity()
```

```java
public class Dashboard extends Controller
{
  public Result index(){
    String email = session().get("email");
    User user = User.findByEmail(email);
    return ok(dashboard_main.render(user.activities));
  }

  public Result uploadActivityForm(){
    return ok(dashboard_uploadactivity.render());
  }

  public Result submitActivity(){
    Form<Activity> boundForm = Form.form(Activity.class).bindFromRequest();
    Activity activity = boundForm.get();

    if(boundForm.hasErrors()) {
      return badRequest();
    }

    String email = session().get("email");
    User user = User.findByEmail(email);
    user.activities.add(activity);
    user.save();
    return redirect (routes.Dashboard.index());
  }
}
```

Dashboard



Activities

Type	Location	Distance
run	tramore	12.0
cycle	dunmore	56.0
walk	fenor	12.0

- Activities list sent to view.
- Scala for loop to iterate over this list, and present in a table.

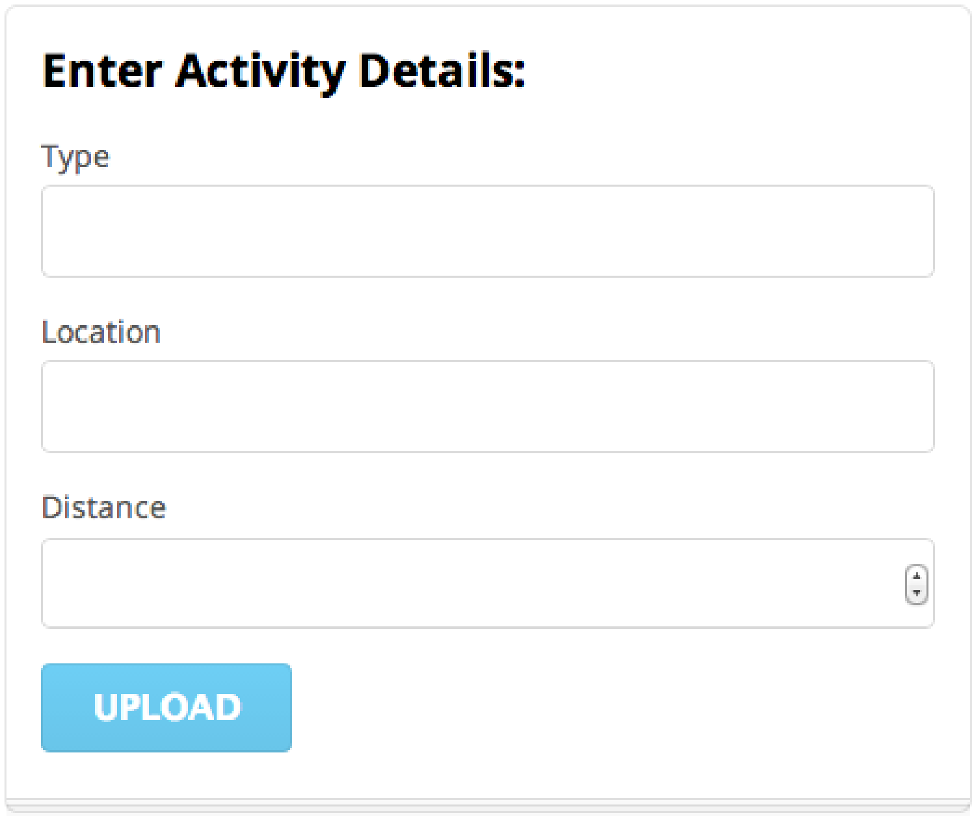
```
public class Dashboard extends Controller
{
  //...
  public Result index()
  {
    String email = session().get("email");
    User user = User.findByEmail(email);
    return ok(dashboard_main.render(user.activities));
  }
  //...
}
```

```
<table class="ui celled table segment">
  <thead>
    <tr>
      <th>Type</th>
      <th>Location</th>
      <th>Distance</th>
    </tr>
  </thead>
  <tbody>
            @for(i <- 0 until activities.size) {
              <tr>
                <td> @activities(i).kind </td> <td> @activities(i).location </td> <td> @activities(i).distance </td>
              </tr>
            }
  </tbody>
</table>
```

dashboard_main.scala.html

```html
<form action="/submitactivity" method="POST">
  <h3 class="ui header">Enter Activity Details: </h3>
  <div class="field">
    <label>Type</label>
    <input type="text" name="kind">
  </div>
  <div class="field">
    <label>Location</label>
    <input type="text" name="location">
  </div>
<div class="field">
    <label>Distance</label>
    <input type="number" name="distance">
  </div>
  <button class="ui blue submit button"> Upload </button>
</form>
```

dashboard_uploadactivity.scala.html



Upload Activity

```java
public class Dashboard extends Controller
{
  //…
  public Result submitActivity(){
     Form<Activity> boundForm
                  = Form.form(Activity.class).bindFromRequest();
    Activity activity = boundForm.get();

    if(boundForm.hasErrors()) {
      return badRequest();
    }

    String email = session().get("email");
    User user = User.findByEmail(email);
    user.activities.add(activity);
    user.save();
    return redirect (routes.Dashboard.index());
  }
}
```

Upload Activity

Acquire the Activity object

Ask the session who is 'logged in'

Add the new Activity to this users activities list

Save the updates

Return back to dashboard

```
public Result submitActivity(){

    Form<Activity> boundForm
       = Form.form(Activity.class).bindFromRequest();
    Activity activity = boundForm.get();

    if(boundForm.hasErrors()) {
      return badRequest();
    }

    String email = session().get("email");
    User user = User.findByEmail(email);

    user.activities.add(activity);


    user.save();


    return redirect (routes.Dashboard.index());
  }
}
```

Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see http://creativecommons.org/licenses/by-nc/3.0/