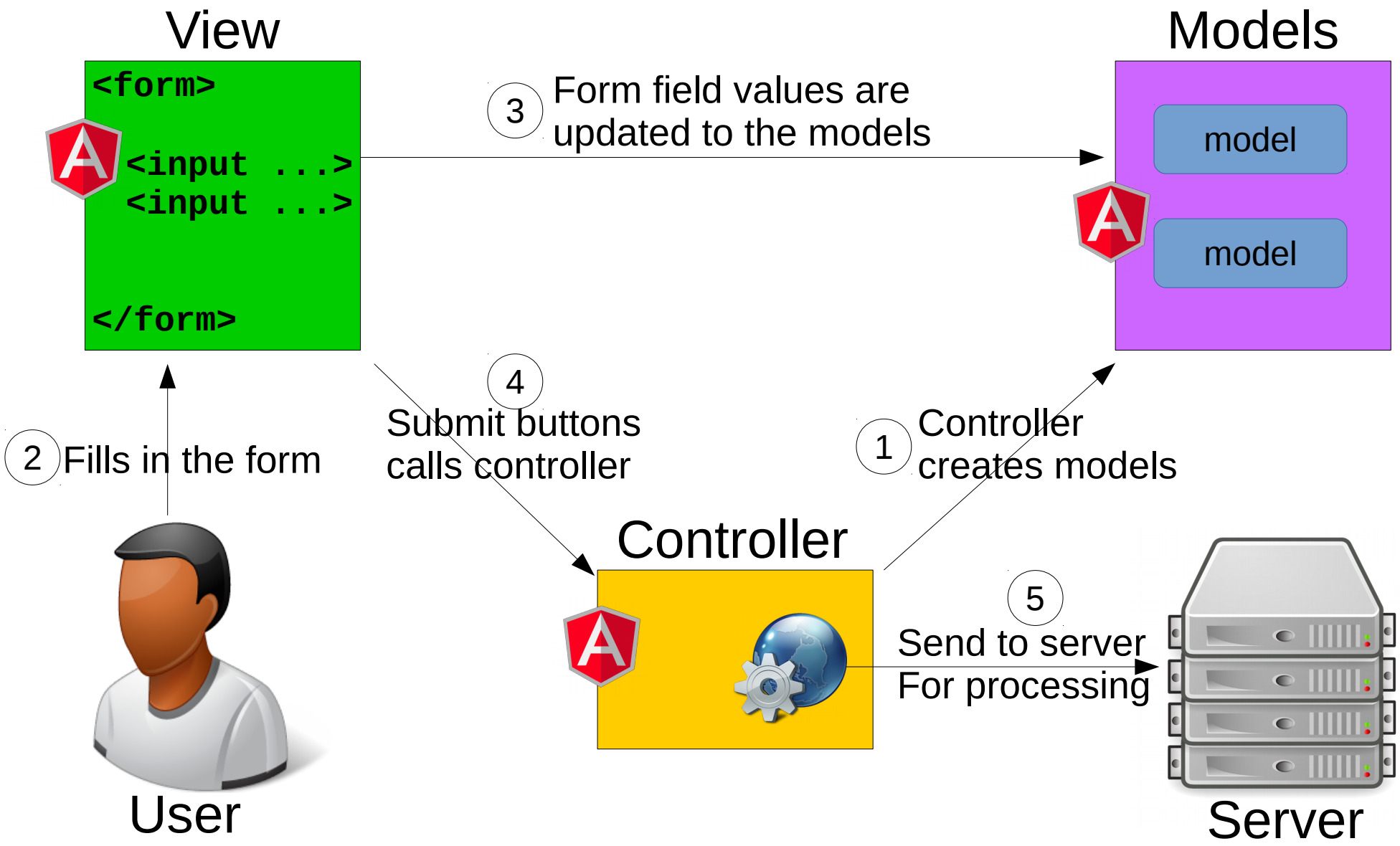
Day 05

Full Stack Foundation

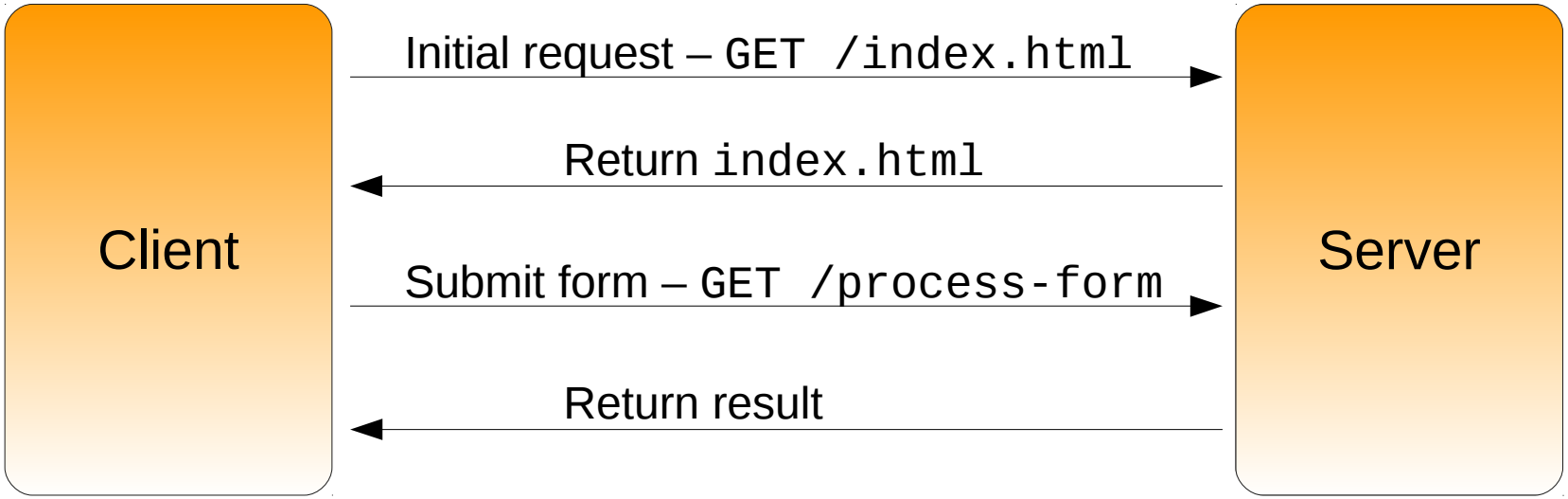
Objectives

- Understanding HTTP protocol
- Processing GET request

Form Processing

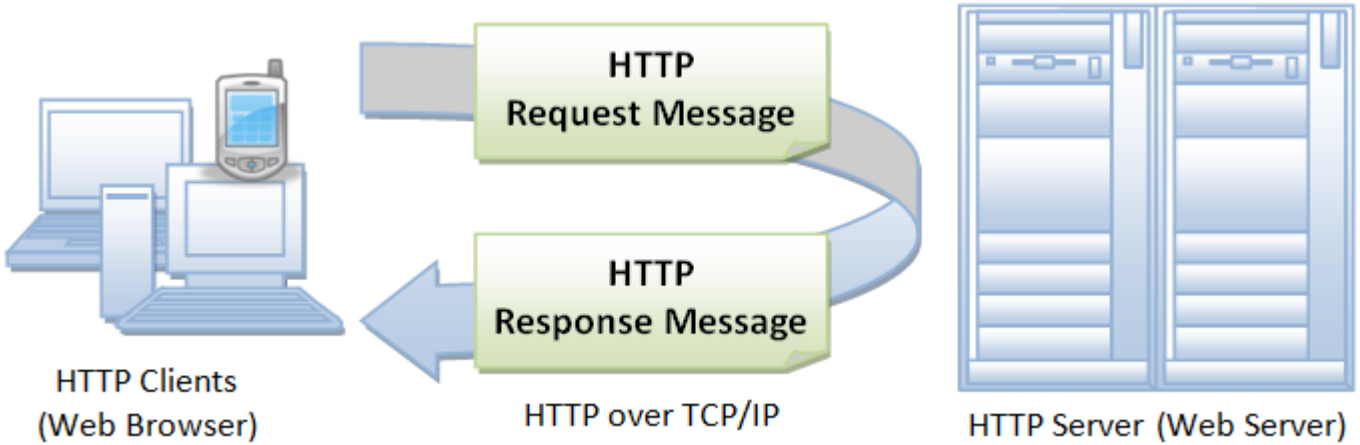


Form Processing



- Initial form is loaded from the server
- User fills in the relevant fields
- When the user clicks the submit button, Angular will take the fields from the form and submits them to the server for processing
  - Using a technique call AJAX
- The server response with the status of the AJAX call
  - Depending on the nature of the submission, the server may return some data to the client
  - The data is encoded in JSON

HTTP Request Response



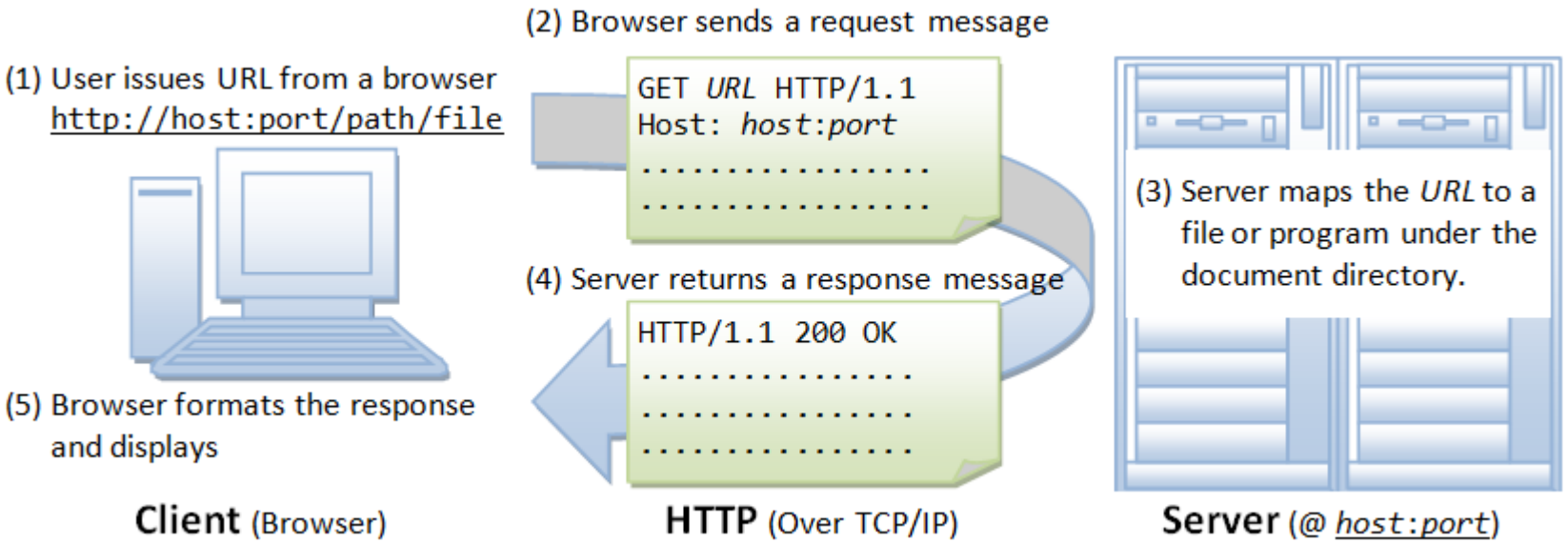
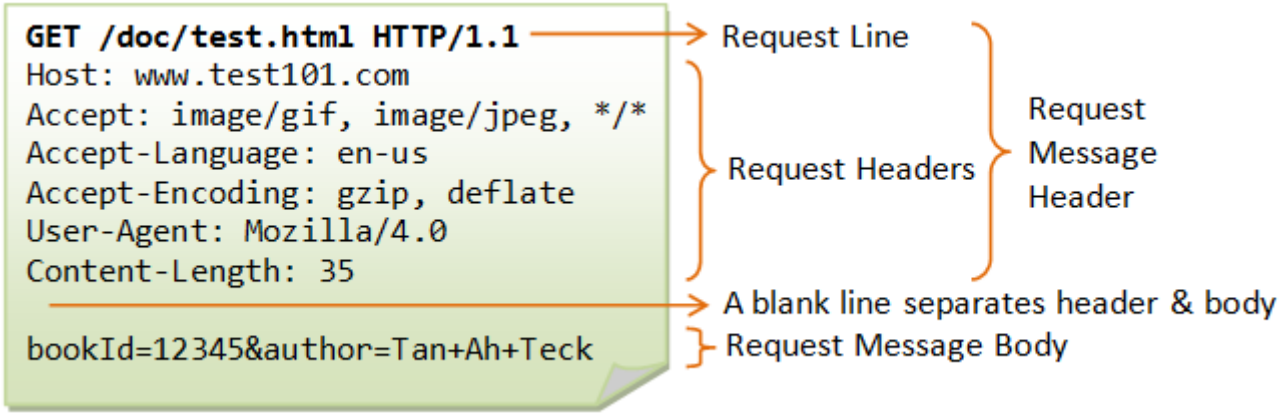


Image from https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

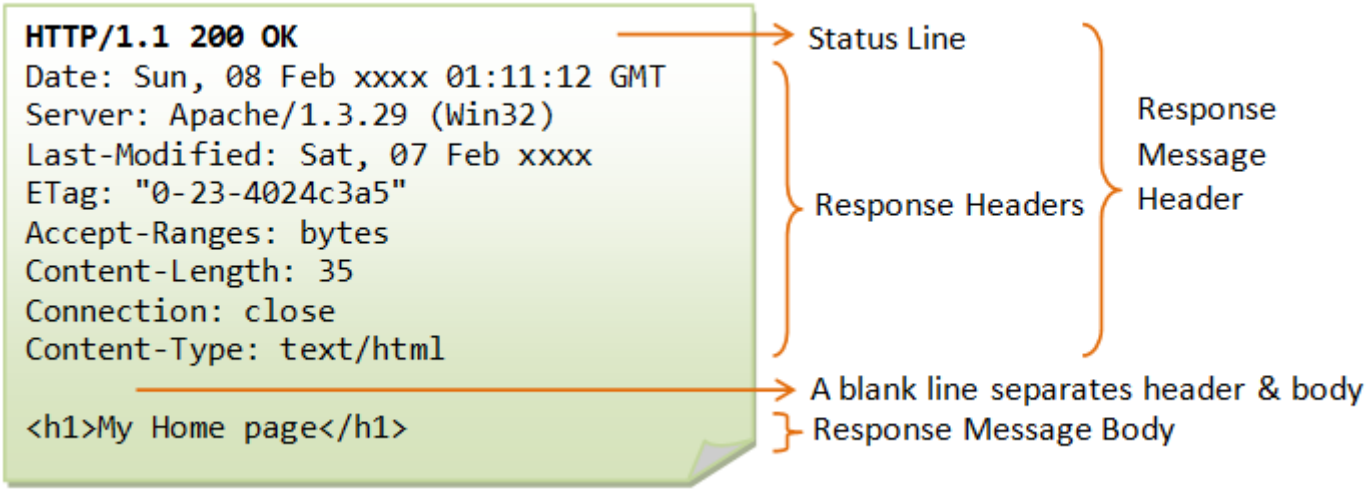
HTTP Request



- Request line
  - HTTP method (GET, POST, PUT, DELETE, etc.) (VERB)
  - Resource – the name of the resource (NOUN)
- HTTP headers
  - Key value pairs
  - Accept – what data format is the client expecting
- Payload
  - Optional
  - If payload is present, then HTTP header will contain Content-Type to indicate to the server what is data format of the payload

Image from https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

HTTP Response



- Response line
  - Status code (100, 200, 300, 400, 500 ranges)
- HTTP headers
  - Similar to Request
- Payload
  - Optional

Image from https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

Submitting Forms with GET

```
<body ng-controller="RegCtrl as ctrl">
<form name="regForm" novalidate ng-submit="ctrl.register()">
    <div class="form-group">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username"
                class="form-control" ng-model="ctrl.username">
    </div>
    <div class="form-group">
        <label for="email">Email:</label>
        <input type="email" id="email" name="email"
                class="form-control" ng-model="ctrl.email">
    </div>
    <div class="form-group">
        <label for="gender">Gender:</label>
        <div class="radio">
            <label class="radio-inline">
                <input type="radio" name="gender"
                        ng-model="ctrl.gender" value="male">Male
            </label>
            <label class="radio-inline">
                <input type="radio" name="gender"
                        ng-model="ctrl.gender" value="female">Female
            </label>
        </div>
    </div>

    <div class="form-group">
        <button type="submit" class="btn-success form-control">Register<button>
    </div>
</form>
</body>
```

Submitting Forms with GET

- `$http` is an Angular service for making AJAX calls to the server
- `$http.get` is used to make a HTTP GET method

```
$http.get("/index.html")

$http.get("/register", { params: {
   username: "fred",
   email: "fred@bedrock.com",
   gender: "male"
});
```

Perform a GET request to this URL

Query string, if any

Query string from the fields in a form

Dependency Injection

Dependency Injection

- Services are created by Angular when they are required by controllers
  - Examples of build-in services: `$http`, `$log`, `$timeout`
- Controllers are said to be dependent on these services
  - Eg. controllers require $http to perform form submission
- These services are passed to the controllers as parameters
  - This process is called dependency injection

Dependency Injection

- Dependents are injected into controller by the "strict-di" method

Array

```
[ "$scope", "$http", "$log", MyController ]
```

One or more services names (dependents)

Controller function

- The services are passed to the controller as parameters when the controller is activated

```
var MyController = function($scope, $http, $log) {

}
```

Submitting Forms with GET

```
(function() {

    var RegApp = angular.module("RegApp", []);

    var RegCtrl = function($http) {
        var ctrl = this;
        ctrl.username = "";
        ctr.email = "";
        ctrl.gender = "";

        ctrl.register = function() {
            $http.get("/register", { params: {
                username: ctrl.usrname,
                email: ctrl.email,
                gender: ctrl.gender
            });
        };
    });

    RegApp.controller("RegCtrl", ["$http", RegCtrl]);
})();
```

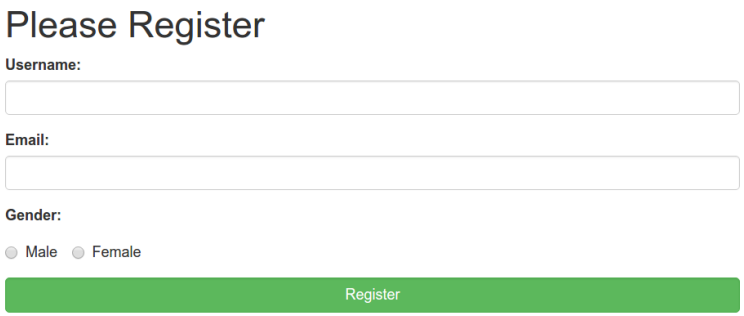
Submitting Forms with GET

```
$http.get("/register", { params: {
   username: ctrl.username,
   email: ctrl.email,
   gender: ctrl.gender
});
```

HTTP **GET**

- GET method is used by the browser to GET a resource from a server
  - When a URL is typed in the address bar
  - When we click on a “link”
- Forms can be submitted using GET method
  - Form fields are encoded as part of the URL – the query string
  - The query string is an optional part of the GET method
  - POST is the other way of submitting form

Submitting Forms with GET



Form

Server

Submit from to `/register`
URL to be processed

Start of query string

```
http://server:3000/register?
username=fred&email=fred@bedrock.com&gender=male
```

Query string

Field name

Field value

Field delimiter

Submitting Forms with GET

```
$http.get("/register", { params: {
    username: ctrl.username,
    email: ctrl.email,
    gender: ctrl.gender
});
```

Generate a HTTP GET method call to the server

```
HTTP/GET /register?
username=fred&email=fred@bedrock.com&
gender=male HTTP/1.1
```

Submitting Forms with GET

- The GET request must be handled with an Express GET middleware
- The fields in the query string appear as properties in req.query
  - The key becomes the property name of req.query
  - Eg `req.query.email` if the query string has a field called email

```
app.get("/register", function(req, res) {
   var username = req.query.username;
   var email = req.query.email;
   var gender = req.query.gender;

});
```

Submitting Forms with GET

```
$http.get("/register", { params: {
    username: ctrl.username,
    email: ctrl.email,
    gender: ctrl.gender
});
```

Generate a HTTP GET method call to the server

```
GET /register?
username=fred&email=fred@bedrock.com&
gender=male HTTP/1.1
```

HTTP GET method is process by the server

express

```
app.get("/register", function(req, res) {
    var username = req.query.username;
    var email = req.query.email;
    var gender = req.query.gender;

});
```

Handling Response

- The server returns a status code once the form fields have been processed
- Used to inform the client as to the status of the submission
  - All HTTP request must have a corresponding response status code
- Status code
  - 2XX – success. Data has been accepted by the server
    - 202 Accepted
  - 4XX – client error. There is an issue with the request
    - 400 Bad Request
  - 5XX – server error. The server runtime environment has experienced a fault
    - 500 – Internal Server Error

Completing the Form Submission

- Must return a status code after processing the form submission request
  - As a general rule, you should always return a status code
- Status code should reflect the processing status of the form
  - Eg. 201 Created – if the server has registered a user
  - Eg. 400 Bad Request – if the form is incomplete

Completing the Form Submission

- Use the response to return the status to the client
  - Send an OK back to the client

```
res.status(200).end();
```

  - Send a Bad Request with a custom message

```
res.status(400).send("Missing email");
```

  - Send a Not Found along with an image

```
res.status(404).sendFile(__dirname +
      "/public/404.png");
```

- A response must end with either `end()`, `send()` or `sendFile()`
  - Otherwise the response is not complete. The client will appear hung
  - Can only call <u>one of the methods once</u>

Completing the Form Submission

express

```
app.get("/register", function(req, res) {
    var username = req.query.username;
    var email = req.query.email;
    var gender = req.query.gender;

    res.status(200).end();
});
```

Generate a HTTP response back to the client

```
HTTP/1.1 200 OK
```

Completing the Form Submission

- Almost all JavaScript I/O functions are asynchronous
- `$http.get()` happens in the background
  - Returns a promise object that will notify the client of the result of the call
    - `then()` - for 2XX status
    - `catch()` - for 4XX and 5XX status
  - Both the method accepts a callback function which will be invoked when the response returns

Completing the Form Submission

```
var promise = $http.get("/register", { params: {
   username: ctrl.username,
   email: ctrl.email,
   gender: ctrl.gender
});

promise.then(function() {

});

promise.catch(function() {

});
```

This callback function will be called if the status code is 2XX

This callback function will be called if the status code is 4XX or 5XX

Completing the Form Submission

express

```
app.get("/register", function(req, res) {
    var username = req.query.username;
    var email = req.query.email;
    var gender = req.query.gender;

    res.status(200).end();
});
```

Generate a HTTP response back to the client

```
HTTP/1.1 200 OK
```

```
$http.get("/register", { pa
    username: ctrl.username,
    email: ctrl.email,
    gender: ctrl.gender
}).then(function() {

}).catch(function() {

});
```

HTTP response is examined. If it is 2XX the then callback function is called. If the status code is 4XX or 5XX, then the catch callback is called