CS 326
REST Web Services

Stephen Freund

Old School (90's)
- Server makes full static web page. Client renders page

The 00's
- Server makes full web page
  - includes some JavaScript for callbacks to server for dynamic update of portions of pages.

The 10's
- Single-page applications. No reloads.
  - JavaScript in browser rewrites web page (DOM)
**REST Mobile Apps**

- Same protocol to talk to server
- App written for native platform
  - iOS, Android
  - not JavaScript in browser

**REST**

- REST == Representational State Transfer
- REST is a convention on top of good old HTTP.
- Basic properties of REST
  - Request/response data format: JSON, xml, etc.
  - Stateless: Don't assume the server knows anything about client state.

**Client/Server Interface**

- The (http) web server performs all the necessarily central actions
- The JavaScript in the browser or the iOS app:
  - presents the data pulled from the server
  - accepts user input, which may prompt it to make a request to server

**Implementing a REST "MicroService"**
Basic REST Requests

- **Type (Method)**
  - GET: get some data
  - POST: put up new data
  - PUT: update or overwrite data
  - DELETE: delete data
- **URL**
  - protocol://host:port/path?query

RESTful Interface Endpoints

- **Resource**: a particular data item on server
  - ex: a "to do" item
- **Route**: is a URL suffix to address a resource
  - All items: /todos
  - Single item: /todos/5
  - Items matching query: /todos?id=5

**Endpoint**: route plus an HTTP method (POST/GET/UPDATE/DELETE/...)
  - eg: GET /todos?id=5

Good REST API Example

To Do List App

Implementing a REST "MicroService"

Data Representations

ToDoItem Model and Endpoints

// Data Model for To Do Items.
class ToDoItem {
    public let id : Int
    public let task : String
    public let done : Bool
    public let created : String
}

ToDoList Endpoints
Data Representations: Manual

JSON → ToDoItem Object

- Slightly painful to encode

```swift
// convert item to JSON string
let str = ""
{ "id": "(item.id)",
  "task": "(item.task)"
  ...
}"
```

Data Representations: Manual

JSON → ToDoItem Object

- Slightly painful to decode

```swift
// convert from JSON Data object for one item
let data = Data(contentsOf: url)
let json = try?
  JSONSerialization.jsonObject(with: data,
  options: [])
let map = json as! [String: Any]
let item = Item(id: map["id"] as! Int,
  task: map["task"] as! String,
  ...
)
```

Data Representations: Codeable Objects

JSON → ToDoItem Object

```swift
class ToDoItem : Codeable { ... }
```

// convert to JSON Data object
let encoder = JSONEncoder()
let data = try? encoder.encode(item)

// convert Data object to String
let json = String(data: data!, encoding: .utf8)

Data Representations: Codeable Objects

JSON → ToDoItem Object

```swift
// convert from JSON Data object for one item
let data = Data(contentsOf: url)
let decoder = JSONDecoder()
let item = try?
  decoder.decode(ToDoItem.self,
  from: data)
```

// convert from JSON Data object for array
```swift
let items = try?
  decoder.decode([ToDoItem].self,
  from: data)
```
Data Representations

- Service's job
  - Objects
  - SQL data

Create SQLite database table
- Each row represents one task
  - Object relational mapping (ORM)
  - Manual mapping (write SQL queries, translate results)
- Use manual mapping until it becomes painful

### ToDoController

```swift
private let server = HttpServer()
private let todos = ToDoService()

func launchService() {
    server.POST["/todos"] = { self.add($0) }
    server.GET["/todos"] = { self.items($0) }
    server.PUT["/todos"] = { self.update($0) }
    server.DELETE["/todos"] = { self.delete($0) }
    server.start(port)
}

func add(request : HttpRequest) -> HttpResponse {
    todos.add(...)
}
```

### ToDoItem Model and Service

```swift
// Data Model for To Do Items.
class ToDoItem {
    public let id : Int
    public let task : String
    public let done : Bool
    public let created : String
}

// To Do Service API.
class ToDoService {
    var items : [ToDoItem]?
    func insert(task: String, done: Bool, created: String) -> Bool
    func update(id : Int, task: String, done: Bool, created: String) -> Bool
    func delete(id: Int) -> Bool
}
```
**SQL Database**

- **Benefits**
  - DB manages data
  - Provides robust, efficient access in uniform way
  - Persistent, fault-tolerant, scalable, multi-user, ...
  - Standard Query Language
- **Data Definition Language (DDL)**
  - Create/alter/delete tables
- **Data Manipulation Language (DML)**
  - Query one or more tables
  - Insert/delete/modify tuples in tables

**Creating a Table**

```sql
CREATE TABLE "todos" (  
"id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,  
"task" TEXT NOT NULL,  
"created" TEXT NOT NULL,  
"done" BOOLEAN NOT NULL 
)
```

<table>
<thead>
<tr>
<th>id</th>
<th>task</th>
<th>done</th>
<th>created</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Walk Wally</td>
<td>true</td>
<td>11/17/18</td>
</tr>
<tr>
<td>5</td>
<td>Feed Wally</td>
<td>false</td>
<td>11/17/18</td>
</tr>
<tr>
<td>6</td>
<td>Grade Labs</td>
<td>true</td>
<td>11/18/18</td>
</tr>
<tr>
<td>8</td>
<td>Walk Wally</td>
<td>false</td>
<td>11/18/18</td>
</tr>
</tbody>
</table>

**Adding a Row to a Table**

```sql
INSERT INTO "todos" ("task", "done", "created")  
VALUES ("Prepare Lecture", false, "11/18/18")
```

<table>
<thead>
<tr>
<th>id</th>
<th>task</th>
<th>done</th>
<th>created</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Walk Wally</td>
<td>true</td>
<td>11/17/18</td>
</tr>
<tr>
<td>5</td>
<td>Feed Wally</td>
<td>false</td>
<td>11/17/18</td>
</tr>
<tr>
<td>6</td>
<td>Grade Labs</td>
<td>true</td>
<td>11/18/18</td>
</tr>
<tr>
<td>8</td>
<td>Walk Wally</td>
<td>false</td>
<td>11/18/18</td>
</tr>
<tr>
<td>9</td>
<td>Prepare Lec</td>
<td>false</td>
<td>11/18/18</td>
</tr>
</tbody>
</table>

**SQL Query**

```sql
SELECT * 
FROM "todos" 
WHERE NOT done
```
**SQL Query**

```sql
SELECT * FROM "todos" WHERE NOT done AND created = '11/18/18'
```

**Result:**

<table>
<thead>
<tr>
<th>id</th>
<th>task</th>
<th>done</th>
<th>created</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Walk Wally</td>
<td>false</td>
<td>11/18/18</td>
</tr>
<tr>
<td>9</td>
<td>Prepare Lec</td>
<td>false</td>
<td>11/18/18</td>
</tr>
</tbody>
</table>

**SQL Query**

```sql
SELECT id, task FROM "todos" WHERE NOT done AND created = '11/18/18'
```

**Result:**

<table>
<thead>
<tr>
<th>id</th>
<th>task</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Walk Wally</td>
</tr>
<tr>
<td>9</td>
<td>Prepare Lec</td>
</tr>
</tbody>
</table>

**SQL Query**

```sql
SELECT * FROM "todos" WHERE id = 8 LIMIT 1
```

**Result:**

<table>
<thead>
<tr>
<th>id</th>
<th>task</th>
<th>done</th>
<th>created</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Walk Wally</td>
<td>false</td>
<td>11/18/18</td>
</tr>
</tbody>
</table>

**Update a Row**

```sql
UPDATE "todos" SET done = true WHERE id = 5
```

**Result:**

<table>
<thead>
<tr>
<th>id</th>
<th>task</th>
<th>done</th>
<th>created</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Feed Wally</td>
<td>true</td>
<td>11/17/18</td>
</tr>
</tbody>
</table>

**SQL Query**

```sql
SELECT id, task FROM "todos" WHERE id = 5
```

**Result:**

<table>
<thead>
<tr>
<th>id</th>
<th>task</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Feed Wally</td>
</tr>
</tbody>
</table>

**Update a Row**

```sql
UPDATE "todos" SET done = false WHERE id = 9
```

**Result:**

<table>
<thead>
<tr>
<th>id</th>
<th>task</th>
<th>done</th>
<th>created</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Prepare Lec</td>
<td>false</td>
<td>11/18/18</td>
</tr>
</tbody>
</table>

**SQL Query**

```sql
SELECT id, task FROM "todos" WHERE id = 9
```

**Result:**

<table>
<thead>
<tr>
<th>id</th>
<th>task</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Prepare Lec</td>
</tr>
</tbody>
</table>

### Delete a Row

```
DELETE FROM "todos"
WHERE id = 5
```

<table>
<thead>
<tr>
<th>id</th>
<th>task</th>
<th>done</th>
<th>created</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Walk Wally</td>
<td>true</td>
<td>11/17/18</td>
</tr>
<tr>
<td>5</td>
<td>Feed Wally</td>
<td>false</td>
<td>11/17/18</td>
</tr>
<tr>
<td>6</td>
<td>Grade Labs</td>
<td>true</td>
<td>11/18/18</td>
</tr>
<tr>
<td>8</td>
<td>Walk Wally</td>
<td>false</td>
<td>11/18/18</td>
</tr>
<tr>
<td>9</td>
<td>Prepare Lec</td>
<td>false</td>
<td>11/18/18</td>
</tr>
</tbody>
</table>

### SQL Bindings

- Can write directly in SQL
  
  - hard to integrate into control logic of code

- Use wrapper around Database engine
  
  - write directly in target language (eg: Swift)

- We'll use SQLite bindings:

  ```swift
  if let row = db.pluck(todos.filter(id == n)) {
    return ToDoItem(id: row[id],
                    task: row[task],
                    done: row[done],
                    created: row[created])
  }```