REMI Database

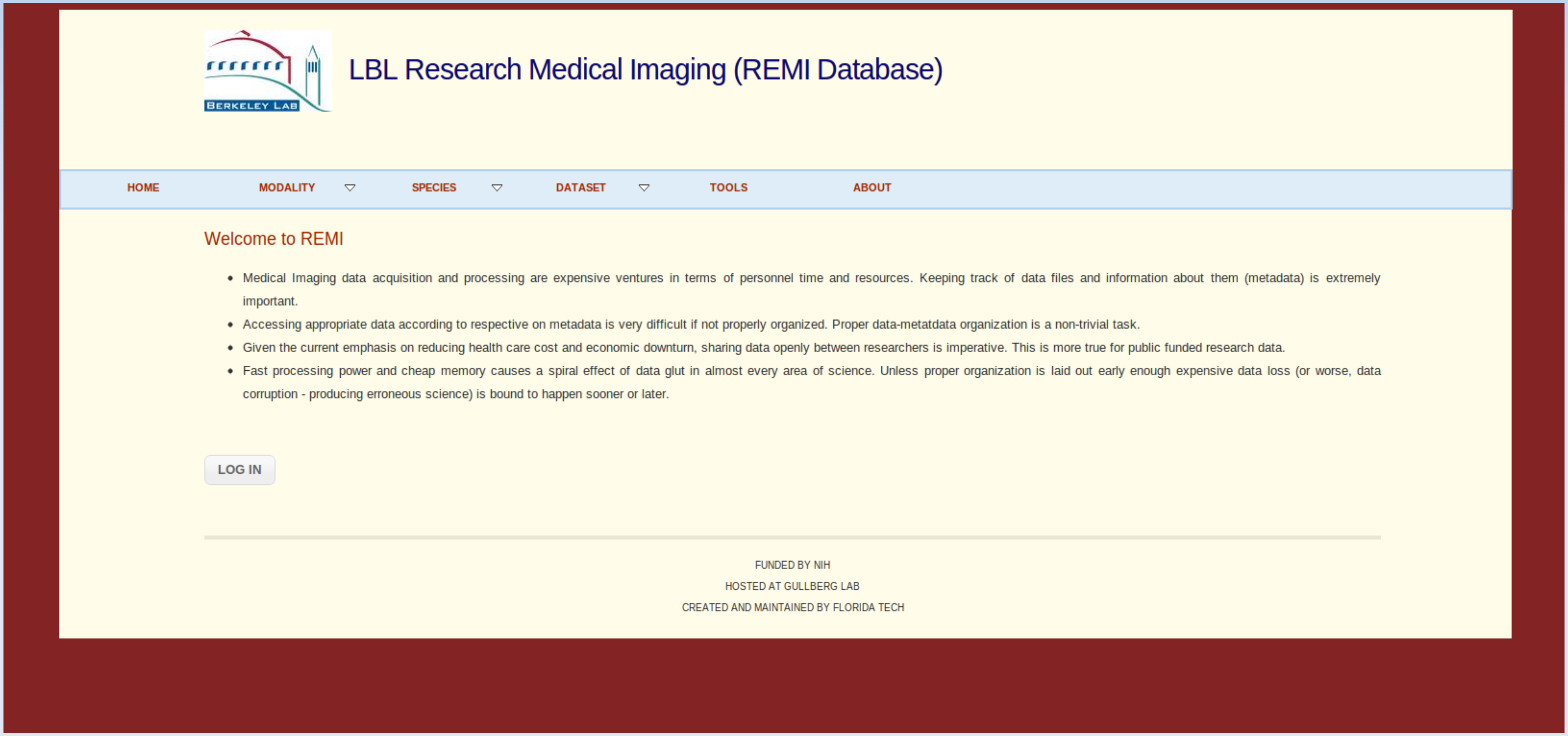
Antall Fernandes

Why we doing what we doing?

Positron emission tomography (PET) machine

Single-photon emission computed tomography (SPECT) machine

REMI (Research Medical Imaging)



LBL Research Medical Imaging (REMI Database)

Welcome to REMI

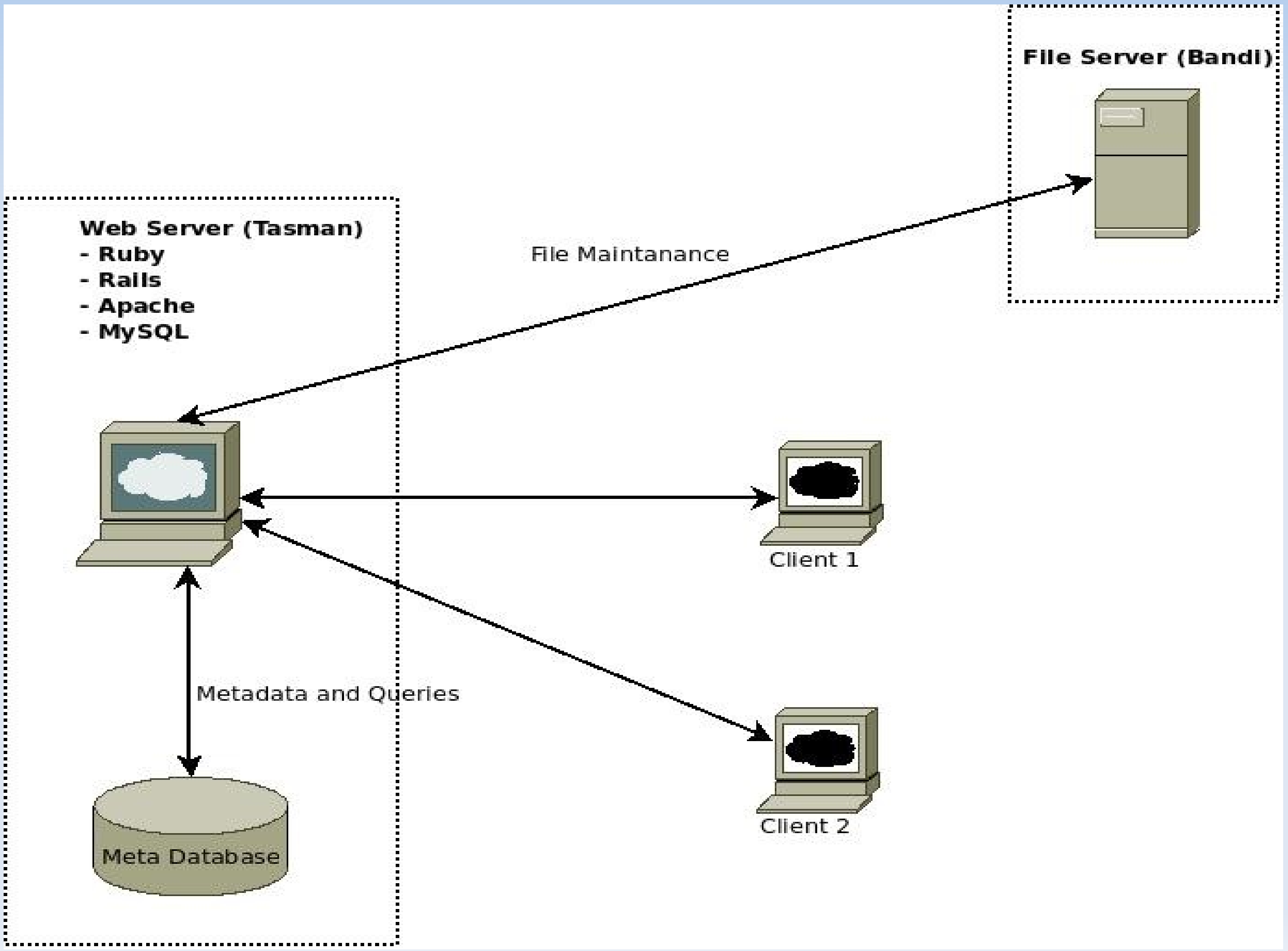
- Medical Imaging data acquisition and processing are expensive ventures in terms of personnel time and resources. Keeping track of data files and information about them (metadata) is extremely important.
- Accessing appropriate data according to respective on metadata is very difficult if not properly organized. Proper data-metatdata organization is a non-trivial task.
- Given the current emphasis on reducing health care cost and economic downturn, sharing data openly between researchers is imperative. This is more true for public funded research data.
- Fast processing power and cheap memory causes a spiral effect of data glut in almost every area of science. Unless proper organization is laid out early enough expensive data loss (or worse, data corruption - producing erroneous science) is bound to happen sooner or later.

FUNDED BY NIH
HOSTED AT GULLBERG LAB
CREATED AND MAINTAINED BY FLORIDA TECH

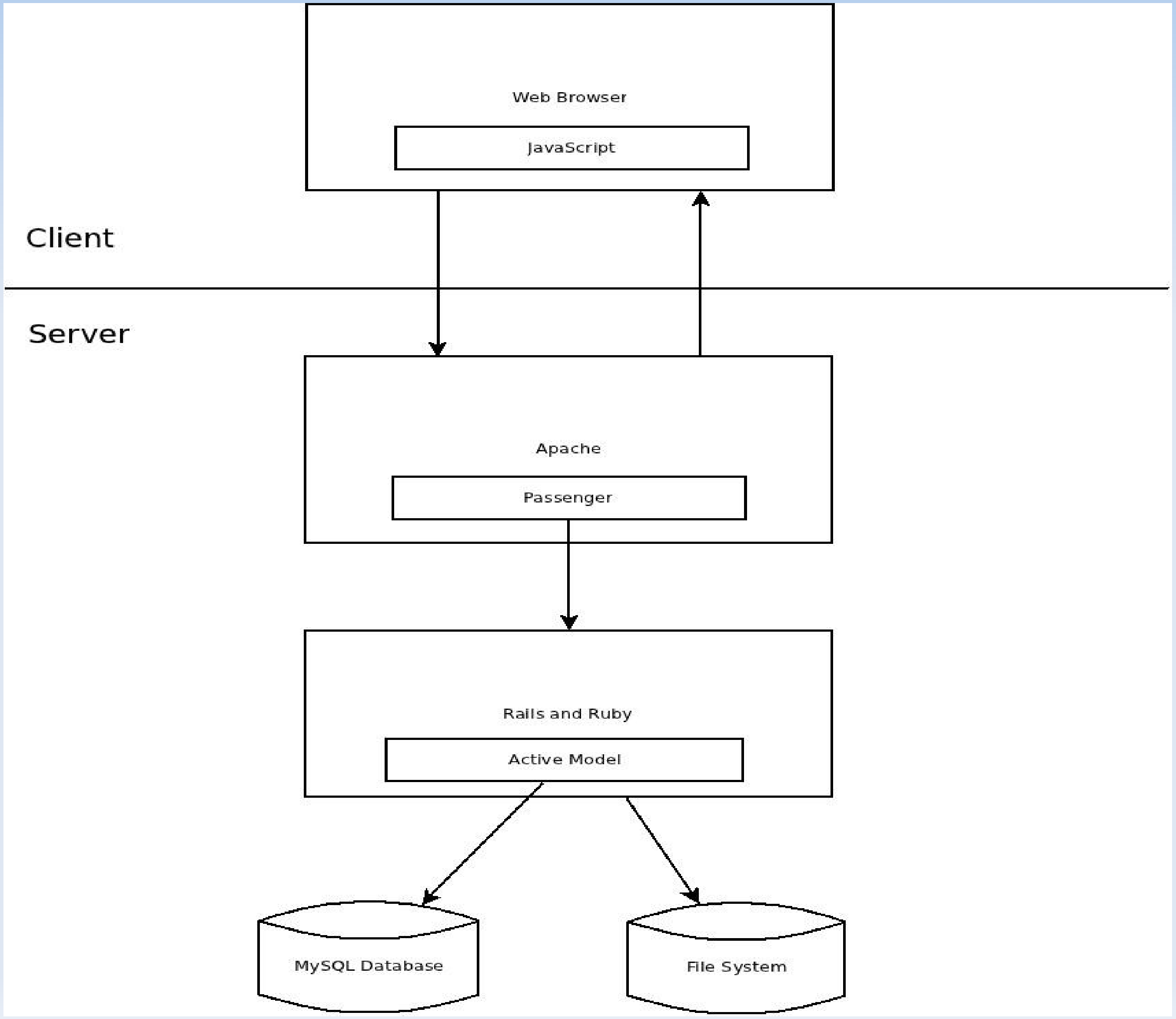
REMI is all about...

- Consolidating study data along with meta data
- Creating of logical collections
- Physical data handling
- Security support
- Data ownership
- Knowledge and information discovery

System Layout



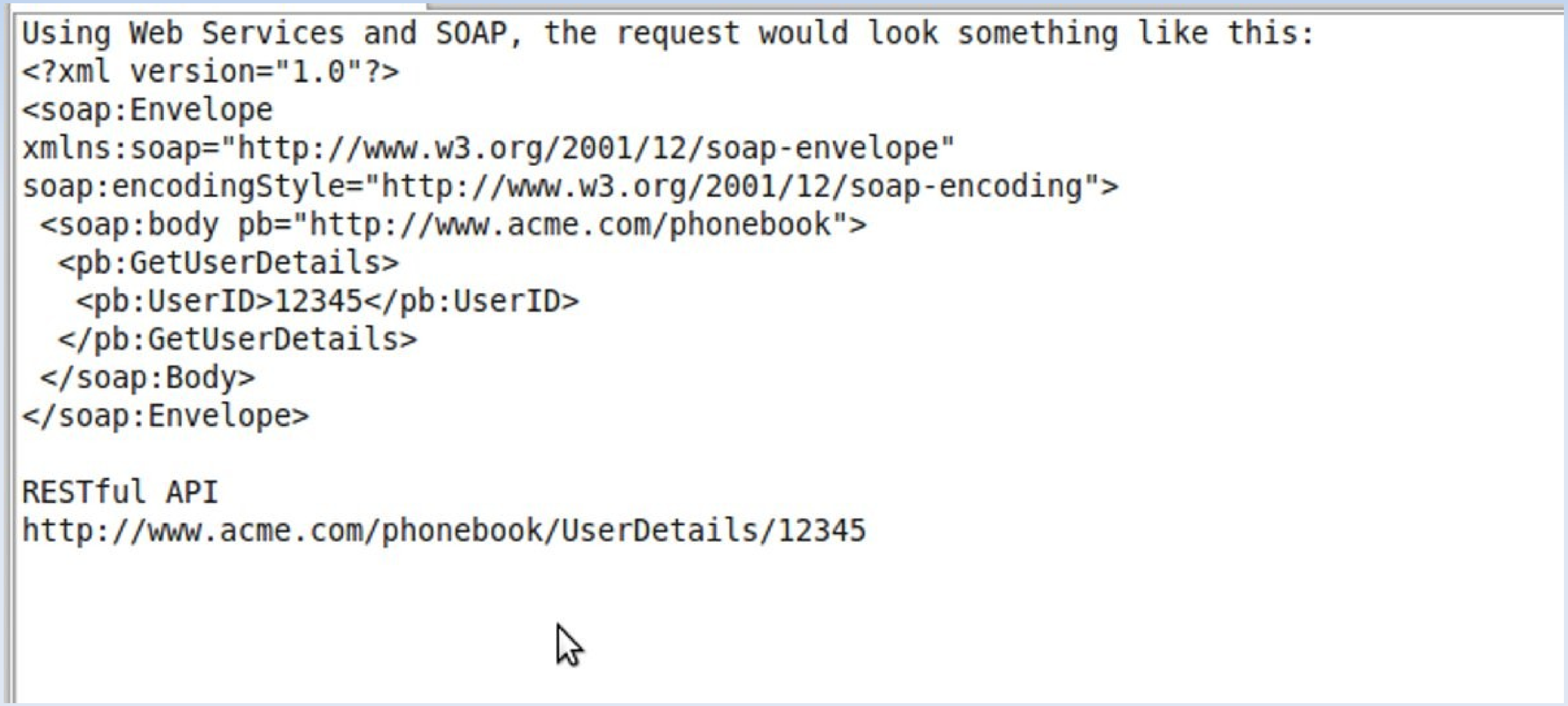
Whats under the REMI hood?



REST (Representational State Transfer)

- REMI follows the RESTful software architecture.
- simple HTTP as opposed to RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL, et al.)
- RESTful applications use HTTP requests to
  - post data (create and/or update)
  - read data (e.g., make queries)
  - delete data.
- REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations.

REST vs SOAP Example



Using Web Services and SOAP, the request would look something like this:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

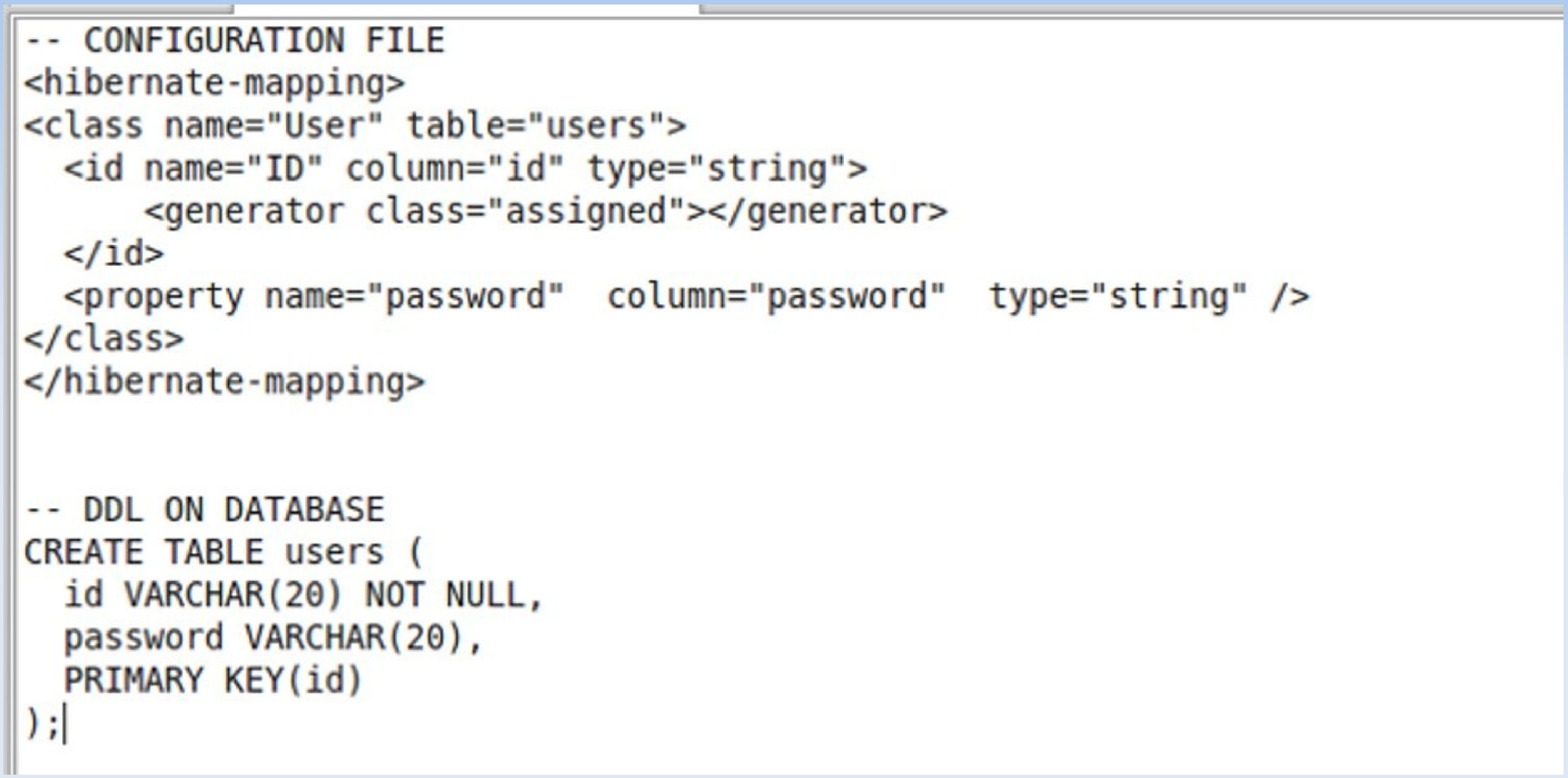
RESTful API

```
http://www.acme.com/phonebook/UserDetails/12345
```

Ruby on Rails (RoR)

- Ruby on Rails uses the Model-View-Controller (MVC) architecture pattern to organize application programming.
- Is intended to emphasize
  - Convention over Configuration (CoC)
  - principle of Don't Repeat Yourself (DRY).
- Utilizes RESTful web services

Configuration File

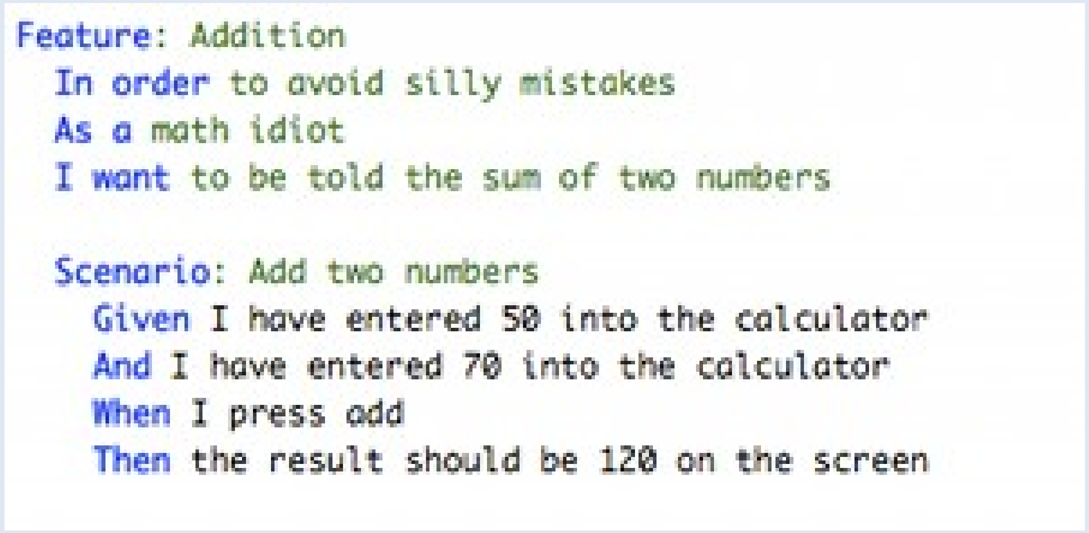


```
-- CONFIGURATION FILE
<hibernate-mapping>
<class name="User" table="users">
  <id name="ID" column="id" type="string">
      <generator class="assigned"></generator>
  </id>
  <property name="password"  column="password"  type="string" />
</class>
</hibernate-mapping>


-- DDL ON DATABASE
CREATE TABLE users (
  id VARCHAR(20) NOT NULL,
  password VARCHAR(20),
  PRIMARY KEY(id)
);
```

Developing REMI

- Behaviour Driven Development (BDD) using Cucumber
  - describe how software should behave in plain text



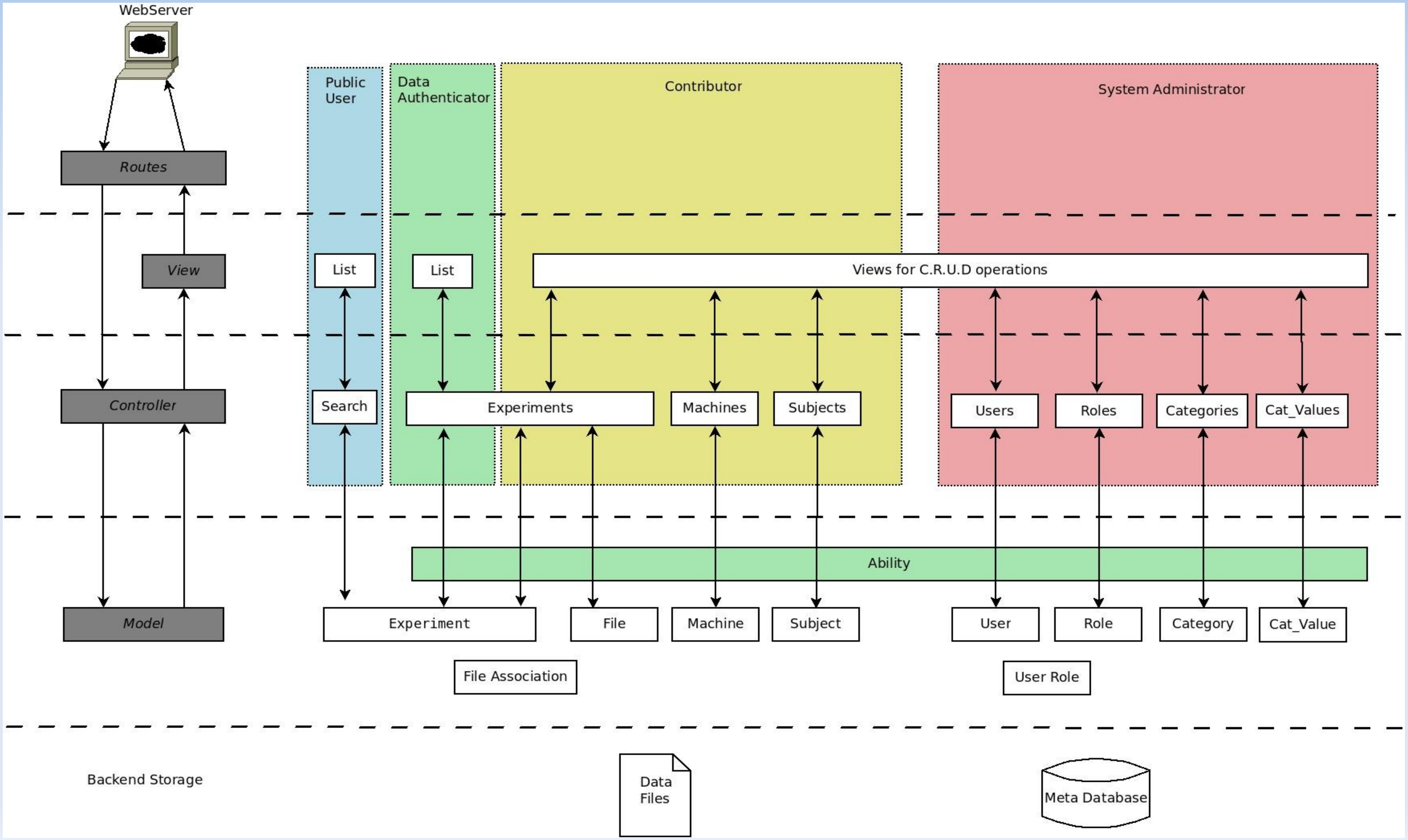
```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario: Add two numbers
    Given I have entered 50 into the calculator
    And I have entered 70 into the calculator
    When I press add
    Then the result should be 120 on the screen
```

Another Cucumber example

```
Feature: pay bill on-line
    In order to reduce the time I spend paying bills
    As a bank customer with a checking account
    I want to pay my bills on-line
Scenario: pay a bill
    Given checking account with $50
    And a payee named Walmart
    And an Walmart bill for $37
    When I pay the Walmart bill
    Then I should have $13 remaining in my checking account
    And the payment of $37 to Acme should be listed in Recent Payments
```

Component/System Design



REMI Test System

Thank You

Future Work

- Validating moving from MRI to JRuby
  - MRI – Ruby execution engine written in C
  - JRuby – Ruby execution engine written in Java
- Move to a dynamic schema design
  - NoSQL could be the way to go
- Handle large downloads more efficiently
- Design a better file upload functionality

Things you will learn...

- Rails
- MySQL
- RSpec
- Cucumber
- Ruby
- Git
- BDD