CANDYGRAM PHP

an api-centric, multi-client, rapid development web framework for php

Andrew Canfield

In partial fulfillment of the requirements for the degree

Master of Science

A thesis presented to

Eastern Washington University

Cheney, Washington

SUMMER 2014

THESIS OF **ANDREW CANFIELD** APPROVED BY

_____ DATE _____

NAME OF CHAIR, GRADUATE STUDY COMMITTEE

_____ DATE _____

NAME OF MEMBER, GRADUATE STUDY COMMITTEE

Table of Contents

Table of Figures

1 INTRODUCTION

1.1 Overview

The web is no longer used as a collection of stateless, static web pages with basic text and graphics. Instead, it has evolved into full web applications serving multiple clients with dynamic content including games, product catalogs, video, email, maps, blogs, and more. This evolution has resulted in web usage growing more prevalent in day-to-day life. Users expect more functionality, interactivity, and interconnectivity between sites as well as the ability to utilize multiple clients including computers, tablets, and phones. As a result, Web development has become more and more complicated. Modern websites requires a developer to be proficient with HTML, CSS, JavaScript, Databases, multiple server-side language such as PHP, C#, Ruby or Python, and any number of application programming interfaces (APIs) and services. The combined challenge of multi-tier, multi-client development has creates code that is inefficient, insecure, and inextensible.

1.2 Lack of Efficiency

Web applications often have similar functionality, but due to the custom nature of each application, this functionality is rarely used interchangeably. Due to the multi-tiered architecture required for web development, code is often repeated between the various layers requiring an object be modeled in JavaScript for the presentation tier, again in a server-side language for the logic tier, and a third time at the database for the data tier [1]. Complicating matters, each layer is implemented using a different modeling and programming paradigms [2, 3, 4].

Server-side languages include PHP, C#, Ruby, Python, Java, and many others. These languages are typically synchronous, linear in execution, and object-oriented, a programming methodology in which "objects" are used to model objects found in the real world [2]. These objects are generated as instances of a class, which is a combination of data and code. This approach allows

a program to be modular, extendable, and more easily understandable. Object-oriented languages are defined as supporting the following principles [2]:

- **Abstraction** - the process of separating ideas from specific instances of those ideas at work.
- **Encapsulation** - enclosing objects in a common interface in a way that makes them interchangeable, and guards their states from invalid changes.
- **Polymorphism** - the ability in computer programming to present the same interface for differing underlying forms (data types).
- **Inheritance** - implements code reuse by allowing an object to be based on another object using the same implementation.

Client-side manipulation of the webpage within the web browsers is considered local, and at this level, utilize JavaScript, an asynchronous, event-driven, prototype-based language where there is no difference between a class and an instance, and polymorphism and inheritance are not supported [5].

Since the web was designed to be stateless—where each request is treated as an independent transaction unrelated to previous requests [6]—data must be persisted at a third layer [1], typically a relational SQL-based database which uses a declarative language where the data is normalized, and then represented in terms of records and grouped into relations [4].

Figures 1, 2, and 3 illustrate these differences by representing a "student" in each paradigm. This student is a person, who may have multiple phone numbers, multiple jobs, and be enrolled in multiple courses:

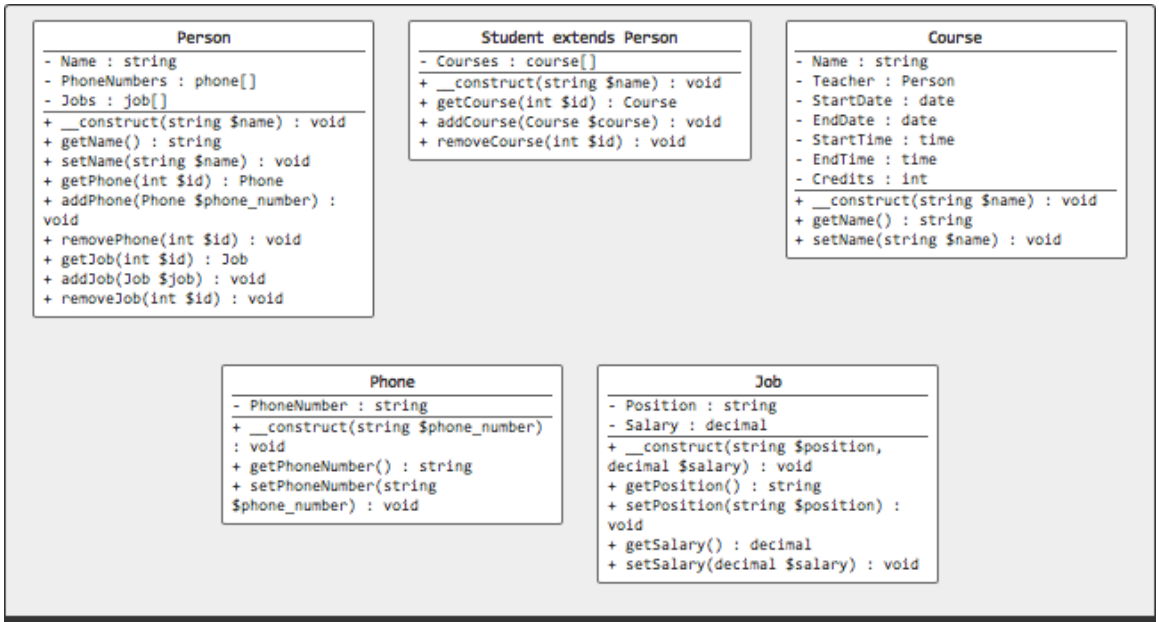

Figure 1: Object Oriented
defined using multiple highly-structured classes

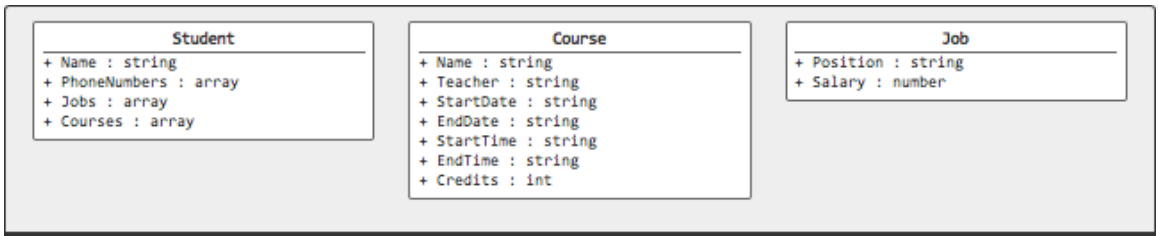

Figure 2: Prototype Based
defined using loosely structured classes

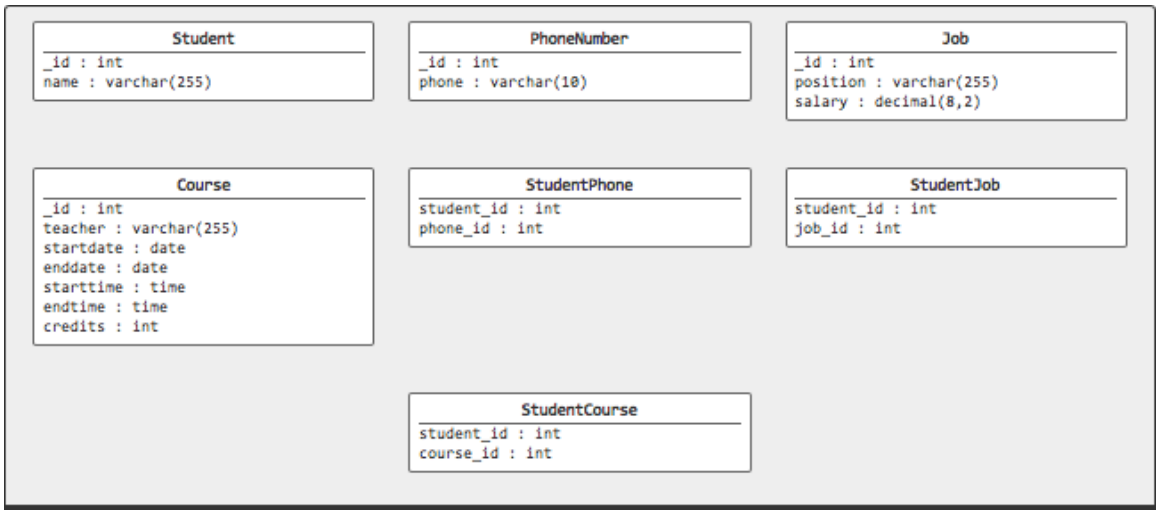

Figure 3: Relational Model
defined using declarative types

Interaction between these layers requires translation from one paradigm to another in the form of Object-Relation-Mapping (ORM), a technique for converting data between incompatible type systems such as object-oriented programming languages and relational databases [7]. Not only is this inefficient, the multiple layers of a web application and the translation create a larger attack surface as multiple vectors and opportunities for bugs are exposed.

1.3 Lack of Security

Web use has grown incredibly since its introduction in the 1990s when 22.9% of households had a computer and average use was 46 minutes a day, compared to 2011 and 2012 where 75.6% of households have a computer and usage had increased to over 4 hours [8, 8]. According to one study, Internet usage is now harder to live without than a landline telephone or television [10]. As society moves to a greater Internet focus, our digital footprint becomes more and more enticing to obtain. Every day, there are news reports regarding the exploitation of major web applications such as Adobe.com, The PlayStation Network, eBay, Twitter, Facebook, Evernote, and many more [11, 12, 13] resulting in over 10 million identities exposed [11]. These attacks can result in the destruction of information, the installation of unwanted software, or the disclosure of confidential user information such as usernames, passwords, and even credit cards [11].

1.4 Lack of Extensibility

As web applications are constantly evolving there is a need for extensibility. Modern applications are no longer hosted as a single web site only accessed using a single browser as a client. Instead, web applications must support multiple clients including desktop browsers, mobile browsers, mobile applications, and even other websites and services as seen with Facebook, Google, Twitter, and many other modern web applications. Each client requires a full code stack specific to the clients' operating system and supported languages, rarely sharing any common components

other than the information stored in the backend database resulting in a vast amount of duplicate code.

1.5 Project Goals

The goal of this project is to create an extensible, rapid-development framework. This framework will provide:

- Support for multiple clients and devices including desktop browsers, mobile browsers, as well as both desktop, tablet, and mobile applications.
- Automatic database generation, eliminating duplicate code.
- Automatically provide create, read, update, and delete (CRUD) functionality at the database level.
- Automatically generated, self-validating forms for list, edit, and detail views at the browser level.
- Common data type components such as text, date, email, and phone numbers that will automate data sanitization and validation preventing many common exploits including XSS and SQL-injection.

2 BACKGROUND

Web development has evolved exponentially from its humble beginnings of intermingled content and presentation consisting of text, images, and hyperlinks, on static HTML pages to a state where modern web pages now require developers to be familiar with multiple languages, paradigms, and engineering practices. It is crucial for a developer to understand the common separations of concerns between front and back-end development; and to be aware of common web vulnerabilities.

Front-end development includes anything that runs on the client, typically but not limited to web browsers; and back-end development is any code that runs on the server. The client accesses the server using a uniform resource identifier (URI)—a compact sequence of characters used to identify an abstract or physical resource [14]. Traditionally, most sites were built using a three-tier architecture with the presentation tier on the front-end, and the logic and data tiers on the back-end [1]. This is no longer the typical development paradigm.

2.1 Front-End Development

With the introduction of HTML5 including the accompanying extensions to cascading style sheets (CSS) and JavaScript, much of the logic, object functionality and manipulation, has joined the presentation on the front-end allowing for real-time interactions. It is even possible to create a data store on the client [15]; however, due to security issues, this is not currently common. Due to these additional responsibilities on the front-end, additional distinctions must be made between HTML, a semantic language used to describe content; CSS, a formatting language used to describe how that content looks; and JavaScript, a programming language for adding behaviors to the content.

Semantic HTML emphasizes the meaning of the encoded information beyond the basic div tag to enable intelligent software agents to automatically find, filter, and correlate information on the web [16, 17]. This has been achieved through additional tags such as: article, section, aside, nav, header, footer, audio, video, additional input types, and output [18].

Along with semantic markup, there has also been strong movement towards utilizing web components that are declarative telling the browsers what to render, such as a date selector, and letting the browser determine how to render the item instead of the previous imperative approach in which the programmer implemented a solution that told the browser what to do [28]. Backward support is provided through a polyfill, downloadable JavaScript code that provides facilities not natively supported by a web browser [29].

CSS has also been greatly extended with new properties and pseudo-classes [19] to permit formatting based on state information; however, it still has certain limitations such as lack of support for variables, meaning if your color theme changes you must change it at every occurrence, and prefixing for non-standardized properties requiring different method signatures for each browser implementation. These and other issues have been resolved by the creation of CSS preprocessors such as LESS and Sass which allow writing clean CSS in a programming constructs, including sub-classing, instead of static rules [20, 21].

JavaScript enables real-time interactions by allowing the client to communicate with the server [5]. This is implemented as the XMLHttpRequest (XHR), also known as an AJAX request, which enables individual parts of a web page to be altered on the fly without a page refresh [22].

Different browsers implement this and other functionality through different methods. To bridge this divide, JavaScript libraries were created to simplify development by providing a common interface and providing additional features. The most common of these libraries is jQuery [23], which is used by over 60% of the 10,000 most visited websites [24]. To minimize bandwidth, these common libraries are stored on content distribution networks (CDN) provided by Google, Microsoft, Amazon, and others [25]. By having a common host, these files may be cached by the browser and used by multiple websites [25].

XHR requests allow data to be interchanged between applications, languages, and even data stores using an application-programming interface (API). Although the X in XHR indicates Extensible Markup Language (XML) is the only supported data exchange format, any format may be used including JavaScript Object Notation (JSON) [22].

XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable [26]. It is a subset of Standard Generalized Markup Language (SGML), a declarative generalized markup language for documents [63], and is the

default format for office productivity tools. XML can us various communication protocols, and can be used to represent almost any data structure. A sample representation of a person object in XML is illustrated in figure 4.

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber type="home">212 555-1234</phoneNumber>
    <phoneNumber type="fax">646 555-4567</phoneNumber>
  </phoneNumbers>
  <gender>
    <type>male</type>
  </gender>
</person>
```

Figure 4: Sample XML representation of a person object

JavaScript Object Notation (JSON) is a lightweight data-interchange format that is both human and machine-readable. It is based on a subset of the JavaScript Programming language and is language independent [27]. JSON consists of objects, arrays, strings, numbers, nulls, and Boolean data types and are represented. A sample representation of the same person object using JSON is illustrated in figure 5.

```
{
    "firstName": "John",
    "lastName": "Smith",
    "isAlive": true,
    "age": 25,
    "height_cm": 167.6,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    },
    "phoneNumbers": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "office",  "number": "646 555-4567" }
    ]
}
```

Figure 5: Sample JSON representation of a person object

2.2 Back-End Development

Just as front-end development has evolved, so has back-end development at both the logic and the data tiers. At the logic tier, there has been an explosion of new frameworks, but most have standardized on an engineering pattern known as Model-View-Controller (MVC).

Model-View-Controller (MVC) separates the modeling of the domain, the presentation, and the actions into three separate entities: the model, which manages the behavior and data of the application; the view, which manages the display of information in the form of output in the appropriate data exchange format; and the controller which interprets the input and output from the user and updating the model and view as required [30].

At the data tier, developers have experimented with new approaches to data storage beyond the typical relational structured query language (SQL) databases. These new database approaches are typically referred to as NoSQL and attempt to solve issues with scale, speed, big data by implementing a non-relation structure and various degrees of ACID—atomicity, consistency, isolation, and durability, compliance [31].

Popular NoSQL approaches include document, graphs, key-value, and column stores which correspond to common data structures [31]. Document stores are similar to objects in that the entire entity is stored in a single schema-less record. Graph stores are similar to multi-linked lists, key-value stores to hashes or dictionaries, and column-stores attribute array. Each approach provides advantages and disadvantages in regards to consistency, availability, and partition tolerance (CAP) [31] and should only be used in the appropriate use cases.

2.3 Common Web Vulnerabilities

Web vulnerabilities typically consist of some form of information disclosure that consists of any attack in which sensitive information is disclosed to an unauthorized user. This can be performed using a cross-site scripting attack, SQL-injection, or other multiple other methods [32].

According to the Open Web Application Security Project (OWASP), inject and cross-site scripting are the top two vulnerabilities on the web for 2013 [33].

SQL-Injection attacks consist of insertion or "injection" of a SQL query via input from the client to the application [34]. A successful SQL injection exploit can read sensitive data from the database, modify database data, execute administration operations on the database, recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system.

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites [35]. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.

Cross-site scripting and SQL injection typically occur when a developer forgets to properly validate and sanitize all user-generated data [32, 33, 34].

3 RELATED WORK

The web development community has attempted to ease development by creating numerous frameworks. These frameworks are designed to aid development of applications by providing common tools and functionality such as database access, templating, and user management [36]. These frameworks promote code reuse by providing additional functionality in the form of plug-ins, extensions, and other additional libraries.

Each framework focuses on a different use cases to determine what functionality is provided, and as a result, there are no set rules regarding what features and functionality are provided. The following section will contain a short summary and evaluation of a sampling of

existing frameworks featuring common approaches such as font-end development, front-end MVC, content management systems (CMS) with plugins, and full-stack server-side MVC.

3.1 Bootstrap

Bootstrap is a front-end framework developed by Twitter used for front-end development of HTML, CSS, and JavaScript by providing common tools such as CSS-preprocessors, JavaScript libraries, web components such as alerts, progress bars, image galleries, and common templates/themes [37]. It is currently the number one project on Github and is used by 3,751,247 websites [38].

Typical usage includes linking to the library through a content delivery network (CDN), and using the supplied CSS styles and JavaScript components to design the interface an application [37].

By standardizing common components, themes, and design, Bootstrap is very efficient for developing web application's front-end. As it does not provide an actual data source, it is not responsible for security; and, by focusing on responsive web design, is extendable to mobile clients. Unfortunately, Twitter only provides front-end functionality and intermixes desktop and mobile clients and does not satisfy this project's goals.

3.2 AngularJS

AngularJS is a front-end MVC JavaScript framework developed by Google used to implement user interfaces using a declarative approach to software components to decouple the manipulating the document object model (DOM)—the browser's internal representation of the web page—from the application logic [39]. The created user interface interacts with a server via XHR requests to obtain data. AngularJS is still relatively new and only used on approximately 50,000 websites; however, it is very popular for use on high usage websites such as YouTube's PS3 application, VEVO, and various Google sites [40]. Typical usage includes linking to the library

through a CDN, linking HTML elements to models and controllers, and then interacting with a server via XHR requests through the controller.

By utilizing declarative models and decoupling the user interface from the application logic AngularJS is very efficient. Furthermore, as it does not provide the actual data source, it does not need to be concerned with security. Finally, it is fairly extensible as it can be re-used for multiple clients. Similar to Bootstrap, AngularJS only provides front-end functionality and does not satisfy this projects goals.

3.3 WordPress

WordPress is a CMS developed in PHP featuring templating, themes, and a plugin architecture extending functionality to include message boards, chat rooms, image galleries, REST servers, e-commerce, and more [41]. Currently it is the most popular blogging software in use on the web powering more than 60 million websites including Time, Google Ventures, Facebook's Newsroom, Sony Music, CBS, CNN, and many others [42].

Typical usage includes uploading the software to a server, linking to a database, selecting a theme, and adding plugins for the desired functionality. This is done through an admin dashboard with little to no actual coding. This allows for very rapid development as well as a thriving eco-system of plugin components [41].

Although plugins are numerous and easy to install, they are not easily customizable for non-standard use cases. In addition, plugins may conflict with each other, and according to a 2013 security analysis by the security firm Checkmarx, seven of the 10 most popular e-commerce plugins and the majority of the top 50 most downloaded plugins are subject to common web attacks such as cross-site scripting and SQL injection [43].

WordPress attempts to resolve these security issues by issuing updates; unfortunately, according to other studies 98% of installations are not up to date either due to a lack of technical ability, or due to the lack of support by required plugins.

Although very convenient for rapid website development, WordPress avoids actual coding, is vulnerable to common web vulnerabilities, and primarily extensible third-party plugins; therefore, it does not satisfy this projects goals.

3.4 Ruby on Rails

Ruby on Rails (RoR) is a full-stack server-side framework developed in Ruby that uses software engineering patterns such as active record, convention over configuration, and MVC [44]. It is one of the most cutting edge development platforms, typically paving the way for other frameworks and introducing new techniques and approaches to the web development community. RoR is used to run more than 600,000 websites including Github, Scribd, Shopify, Hulu, Groupon and was used for the original implementation of Twitter [45]. Additional functionality is available through custom code as well as gems, Ruby's package manager used to distribute programs and libraries.

RoR is opinionated software. It makes the assumption that there is a best way to do things. However, if "The Rails Way" is followed, a developer is able to write less code while accomplishing more than most other frameworks. Typical usage includes installing the Ruby Version Manager (RVM) which allows you to use multiple versions of Ruby, installing ruby using the RVM, installation ruby gems, installing RoR using ruby gems, using RoR to create a new project, linking to a database, then coding routes, controllers, models, views, and templates [44].

Common functionality is provided by the framework; however, the bulk of development is performed by the developer which means Ruby on Rails can be used for nearly any use case including RESTful APIs. In addition, Rails protects against common web attacks such as SQL

injection and Cross-site scripting. The base framework does not provide authentication and authorization, however they are available as gems. And although Ruby on Rails is primarily targeted at a single web-based client, it can be extended to mobile devices using CSS.

Although RoR is efficient, secure, and extensible it still has flaws. Ruby on Rails is complicated and difficult to learn, and even worse, it is in active development and is more than willing to completely change any component based on new best practices or the development community's whims. During development, RoR has moved from the Prototype.js JavaScript Library, to the jQuery JavaScript Library, to CoffeeScript a replacement for JavaScript. Although it satisfies the majority of this project's goals, it is difficult for programmers to pick-up quickly and even more difficult to stay current with best practices.

4 DESIGN

This project will utilize an example of a blog website implemented with PHP as the server-side language. This blog will consist of multiple authors, topics, tags, and user comments. Current non-framework, development techniques will be discussed [46, 47]; then the new paradigm of the streamlined framework will be examined.

4.1 Current Technique

1. PHP is used to code the necessary classes, also known as models, and their appropriate methods. This example includes, author, article, and comment classes each with methods for creating a new object, finding an existing object, and methods to retrieve and

update each attribute in the object. This is illustrated in figure 6.

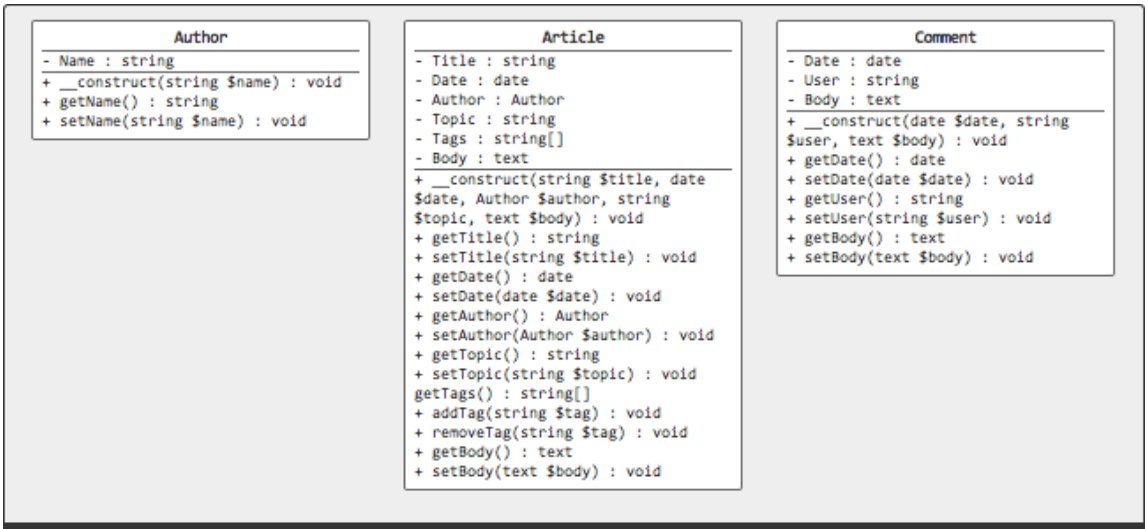

Figure 6: Class UML
Create and manipulate data at the server

2. SQL is then used to create the appropriate, normalized, database structures for authors, articles, comments, topics, and tags as illustrated in figure 7.

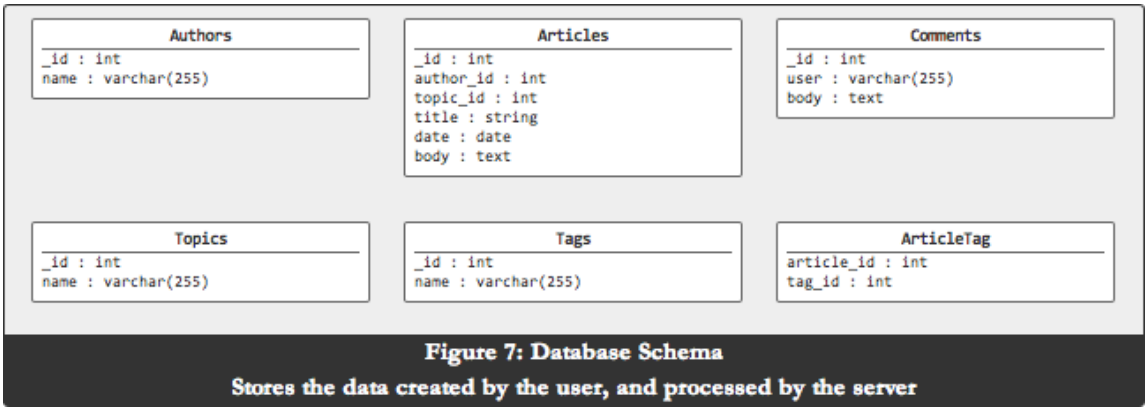

Figure 7: Database Schema
Stores the data created by the user, and processed by the server

3. HTML and CSS are used to create and style the new, edit, list, and detail views for authors, articles, and comments as illustrated in figure 8.

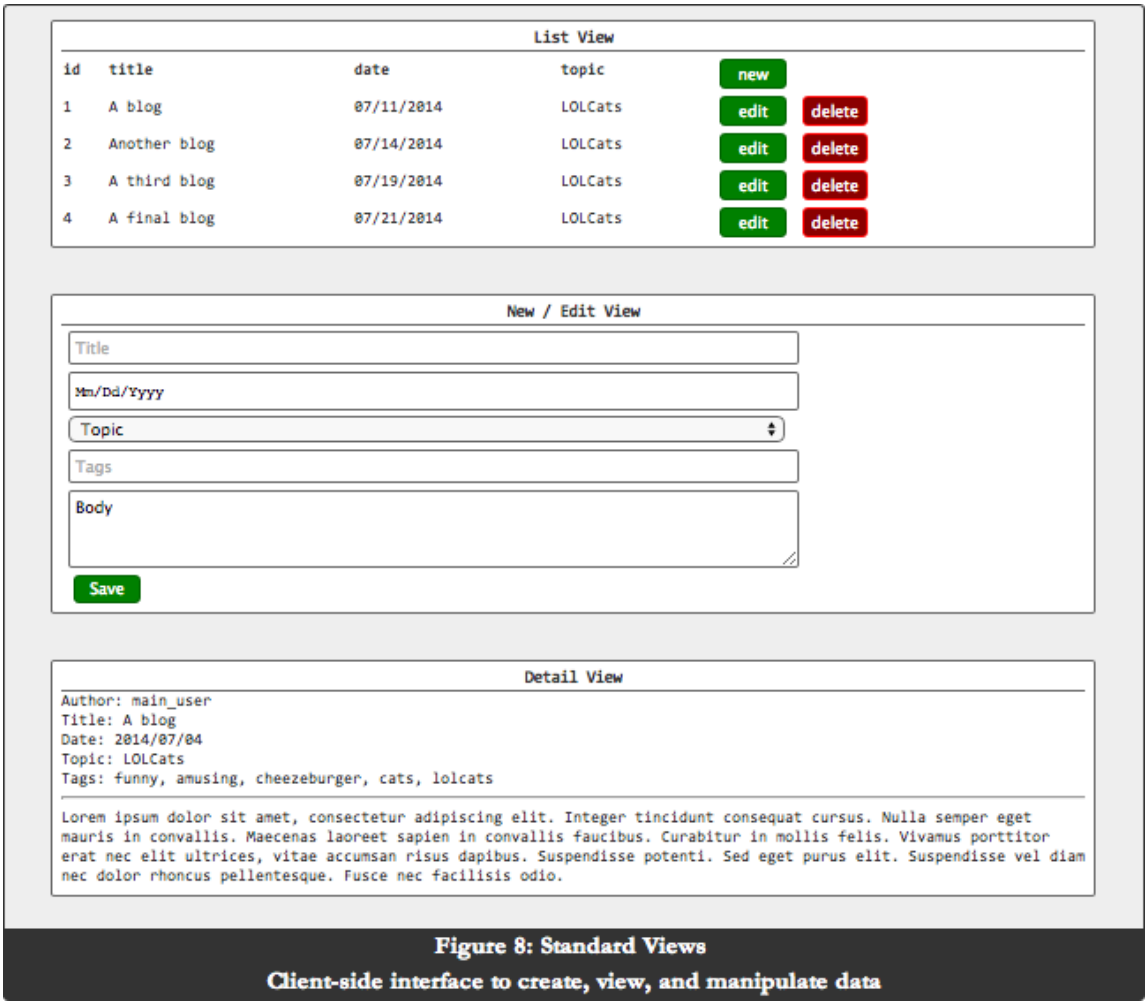

Figure 8: Standard Views
Client-side interface to create, view, and manipulate data

4. JavaScript is used for client-side validation of input as illustrated in figure 9.

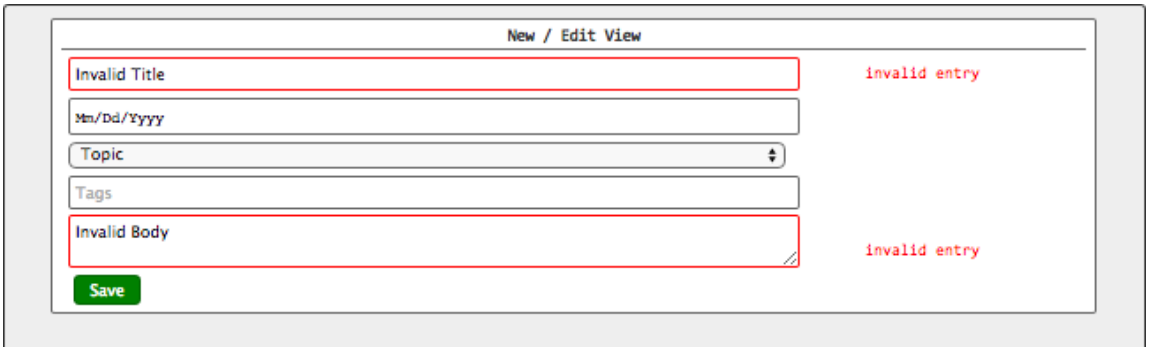

Figure 9: Javascript Validation
provides feedback to user prior to form submission

5. PHP is again used to code the necessary controllers to handle form data, verify validation, and interact with the database.

6. If there is any change to the original model, all code and databases must be updated in a similar process and if functionality is extended to an additional client such as a mobile browser, a similar process will be repeated for each client.

As previously discussed, this development technique is not ideal; it has multiple instances of repetitive code, a large surface for attacks, multiple development paradigms to contend with, and is not easily extendable.

5 Proposed Technique

1. PHP is used to construct a declarative model using standardized fields components such as text, date, email, and phone numbers as illustrated in figure 10.

```
class Blog extends Model {
  protected $author;
  protected $title;
  protected $date;
  protected $tags;
  protected $post;
  protected $comments;

  public function __construct() {
    $this->title = new String();
    $this->title->setProperty('required', true);

    $this->date = new Date();
    $this->date->setProperty('required', true);

    $this->author = new String();
    $this->author->setProperty('required', true);

    $this->topic = new String();
    $this->topic->setProperty('required', true);

    $this->body = new Text();
    $this->body->setProperty('required', true);

    $this->tags = array();
    $this->comments = array();
  }
}
```

```
class Comment extends Model {

  protected $author;
  protected $date;
  protected $comment;

  public function __construct() {
    $this->author = new String();
    $this->author->setProperty('required', true);

    $this->date = new Date();
    $this->date->setProperty('required', true);

    $this->comment = new Text();
    $this->comment->setProperty('required', true);
  }

}
```

Figure 10: Declarative Model

2. The database is automatically generated based on the model.

3. New, edit, list, and detail views are automatically generated using the JavaScript client. These views are injected into the HTML and then styled with CSS.

4. The JavaScript client automatically generates client-side validation.

5. A default controller is provided by the framework for CRUD functionality., and models with more complicated structures can extend the default controller as necessary.

6. If there is any change to the original model, it is automatically implemented into the controller and the database. If functionality is extended to an additional client such as a mobile browser, only custom views need to be coded.

This proposed technique provides a simplified, efficient, development while ensuring a reduced attack surface, input sanitation and validation, well-developed common components, and is easily extended to additional clients without having to modify the original client or developing a second class API as an after-thought.

6 METHODOLOGY

6.1 Multi-Client

Modern web applications are no longer limited to a single client interface or device such as a web browser; instead, most applications must support a web browser, mobile browsers, and mobile applications for each mobile OS, and secondary services such as other websites [48].

According to current techniques, each client supported requires a full implementation only sharing a common data store as illustrated in figure 11. Each of these implementations must be updated separately which results in duplicate work and a lack of consistency in both interface and functionality. Furthermore, additional client support, often an afterthought, requires additional work on the original implementation allowing additional clients to access the common data store.

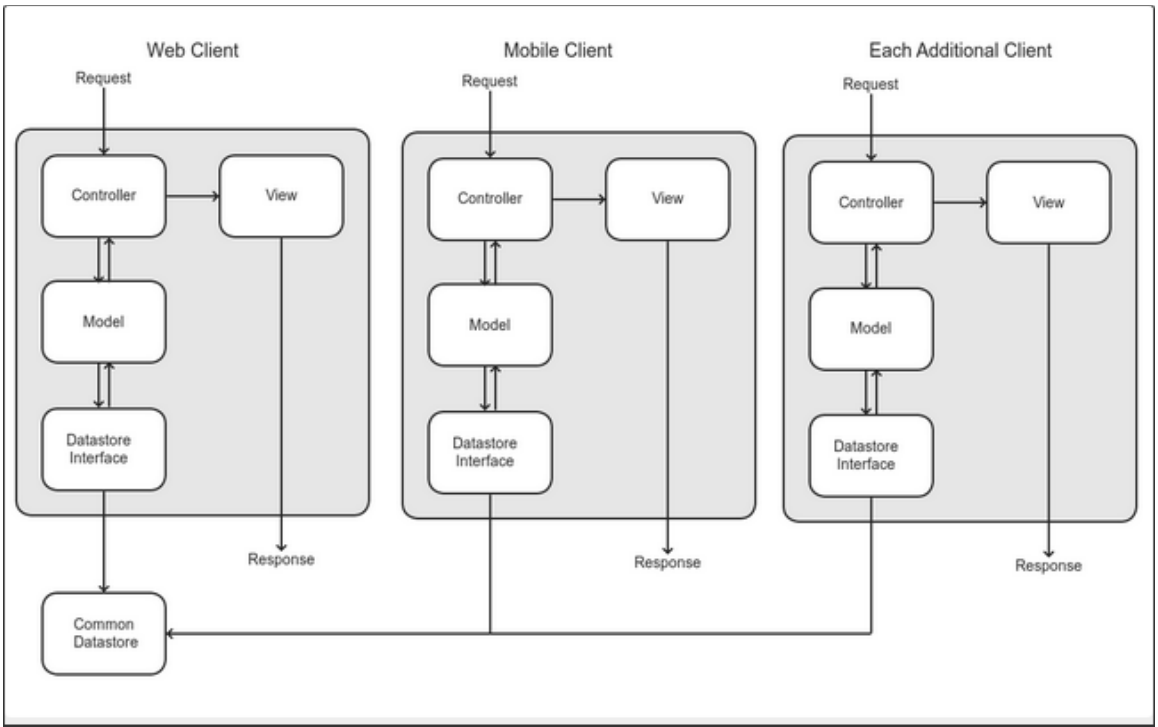

Figure 11: Duplicate Client Code

To remedy this duplication of code, the proposed technique decouples common functionality from client-specific functionality and presentation resulting in API-centric development as illustrated in figure 12. This approach allows greater use of shared code resulting in less duplicate work and providing greater consistency in functionality. There are no longer first and secondary citizens, if the API supports it, so can all the clients. If a new client is needed, no change is required to the original client as it is just another, separate API-subscriber. Each new client only needs to access the API through the standard REST interface and focus on presentation instead functionality resulting in rapider development.

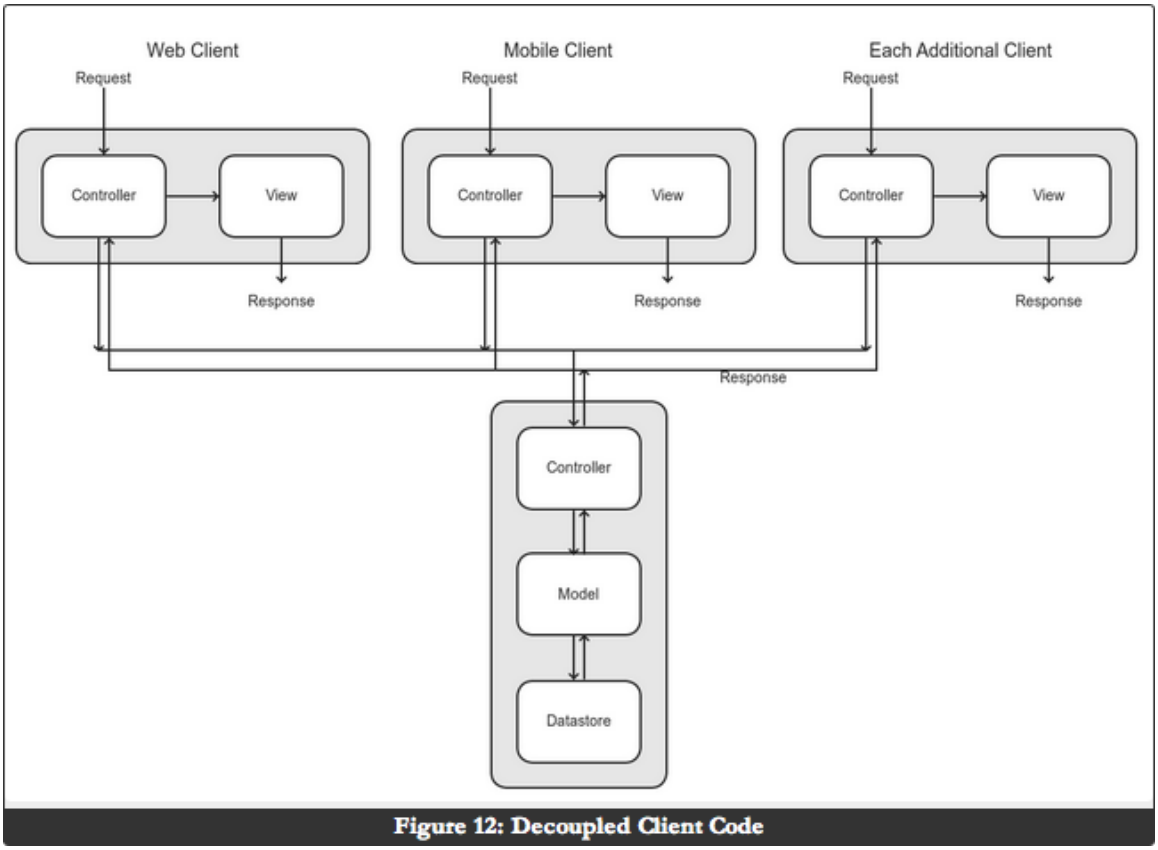

Figure 12: Decoupled Client Code

6.2 API Implementation

There are two competing approach for implementing APIs, Simple Object Access Protocol (SOAP) and Representative State Transfer (REST).

6.2.1 Simple Object Access Protocol (SOAP)

Simple Object Access Protocol is a specification for exchanging XML-based information consisting of three parts: an envelope which defines what is in the message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing procedure calls and responses. SOAP is highly extensible, it can be used over any transport protocol such as HTTP, SMTP, TCP, etc., and it allows for any programming model. Due to its versatility and extensibility, there is no common standard for SOAP interfaces; instead, each domain is responsible to design its own implementation [49, 50].

6.2.2 Representative State Transfer (REST)

Representative State Transfer ignores the details of component implementation and protocol syntax in order to focus on the standard Create, Read, Update, and Delete (CRUD) functionality through a standardized interface using existing HTTP Methods and allows responses to be returned in multiple encodings including XML, JavaScript Object Notation (JSON), HTML, text, etc. REST is the standard interface for most modern web-based APIs [51, 52, 53].

6.3 Summary

In summary, SOAP is used to expose application logic, whereas REST is focused on accessing named resources through a consistent interface using existing HTTP methods. Due to its standardized interface and focus on data access, this project implements its API using a RESTful architecture.

1. **HTTP Methods**

The **GET** method retrieve whatever information, in the form of an entity, is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data that shall be returned as the entity in the response and not the source text of the process. This method is idempotent—can be repeated without changing or otherwise affecting the data [54]. In a RESTful API, GET is used to read an object or search for an existing object [51, 52, 53].

The **POST** method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to annotate existing resources, provide a block of data to a data handling process, and/or extend a database through an append operation [54]. This method is non-idempotent, meaning it may have side effects. In a RESTful API, POST is used to create a new object [51, 52, 53].

The **PUT** method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not refer to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI [54]. This method is idempotent and does not result in any side effects. In a RESTful API, PUT is used to update an existing object [51, 52, 53].

The **DELETE** method requests that the origin server delete the resource identified by the Request-URI. This method is idempotent and does not result in any side effects [54]. In a RESTful API, DELETE is used to delete an existing object [51, 52, 53].

The **OPTIONS** method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating resource retrieval. This method is idempotent and does not result in any side effects [54]. In a RESTful API, OPTIONS is not typically used; however, in the proposed framework, OPTIONS is used to request information about an entity such as type, validations, etc. for the purpose of automating client side functionality.

6.3.1 API Request and Responses

An API request will consist of a URI and an HTTP method. Optionally, a query string or form data may also be provided. Responses will be encoded in JSON and include component information such as name, value, type, and validation as illustrated in figure 13.

```
URI:    http://www.example.com/blog
METHOD: OPTIONS
```


```
{
    "status": "success",
    "status_code": "200",
    "method": "OPTIONS",
    "data": {
        "author": {
            "value": null,
            "required": true,
            "validation": "^(\\w|\\d|\\s){1,255}$",
            "class": "String"
        },
        "title": {
            "value": null,
            "required": true,
            "validation": "^(\\w|\\d|\\s){1,255}$",
            "class": "String"
        },
        "date": {
            "value": null,
            "required": true,
            "validation": "^\\d{4}-\\d{2}-\\d{2}$",
            "class": "Date"
        },
        "brian": {
            "value": null,
            "required": false,
            "validation": "^(\\w|\\d|\\s){1,255}$",
            "class": "String"
        },
        "post": {
            "value": null,
            "required": true,
            "validation": "^.*$",
            "class": "TextArea"
        },
        "comments": [

        ],
        "_id": null,
        "groups": [

        ],
        "permission": "private",
        "class": "Blog"
    }
}
```


Figure 13: Sample API requests and responses

6.4 Routing

Although the API and clients are separate entities, they are able to coexist on the same server as illustrated in figure 14. This is achieved by utilizing apache's mod_rewrite module. This module also allows requests to be split based on which HTTP methods are being used, what headers are being transmitted, and what user agent is present.

Mod_rewrite also allows the use of clean URIs, and the reformatting of incoming URIs to prevent malformed requests from accessing the application.

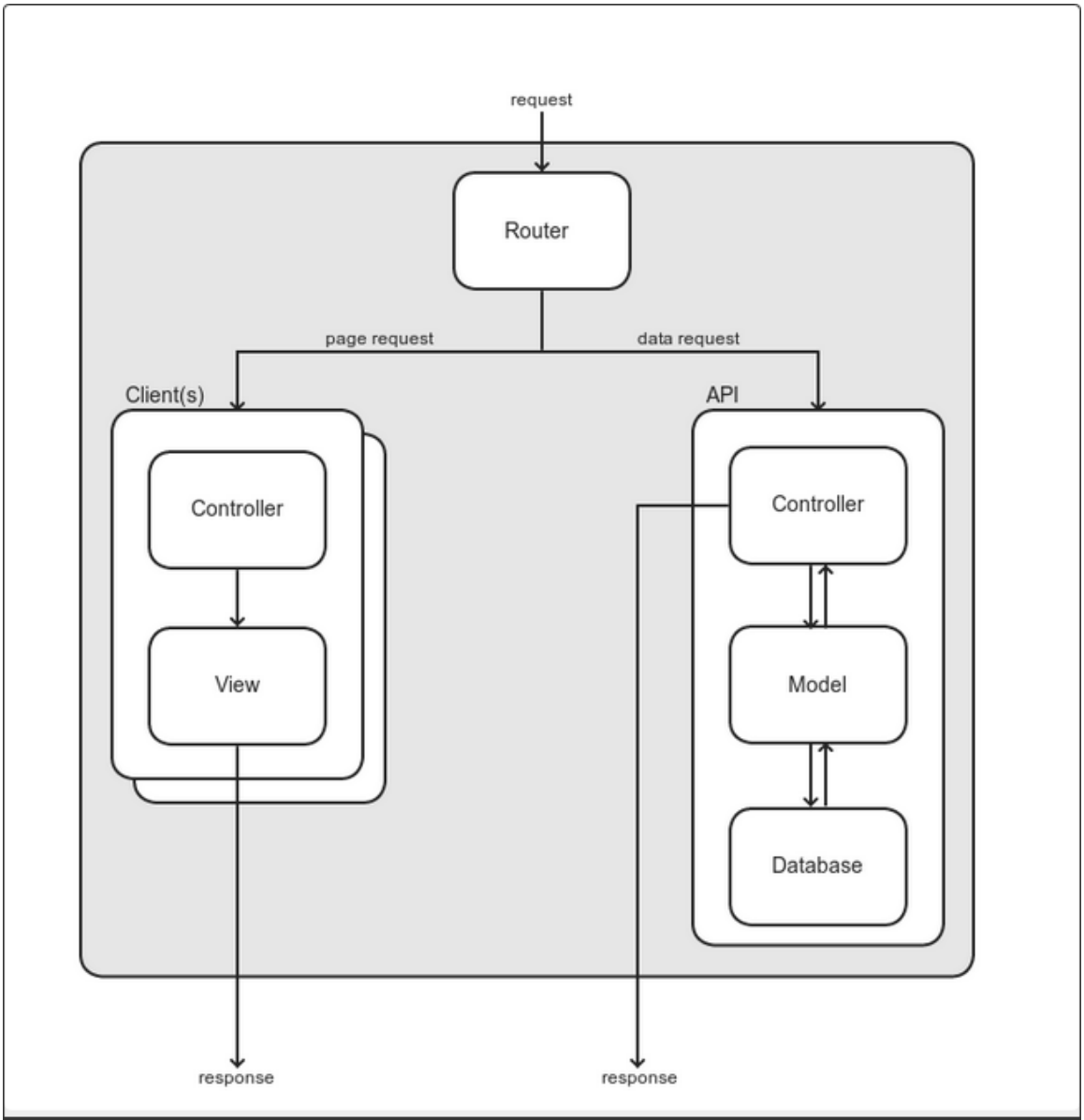

Figure 14: Routing to allow shared server

6.4.1 Apache's Mod_rewrite Module

Mod_rewrite can be used to generate clean URIs, prevent image hot-linking, ensure SSL, establish canonical host names, provided custom 404s, act as a proxy server, and much more [55, 56].

Mod_rewrite examines incoming URI requests, compares them against rules and conditions consisting of server variables, regular expressions, and flags, then rewrites and resubmits the URI based on the rules and conditions provided. These rules and conditions are specified in the .htaccess file that is located in the root web directory. The rules and conditions apply to all child-directories although behaviors can be modified with additional .htaccess files in these child-directories [55, 56].

Regular Expressions were created in 1956 by the American Mathematician Stephen Cole Kleene, and are a sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations. These patterns consist of literal characters and meta-character classes such as words, digits, and white spaces as illustrated in figure 15. These characters and classes can then be modified through grouping, alternatives using or, or set as a range using the square brackets. They can then be quantified as required, optional, zero or more, one or more, or any specific range and finally anchored to the start of the line, or the end of it [57]. Together, these characters, classes, modifiers, and anchors can be used to identify textual material of a given pattern, or process a number of instances of it that can vary from a precise equality to a very general similarity of the pattern as illustrated in figure 16.

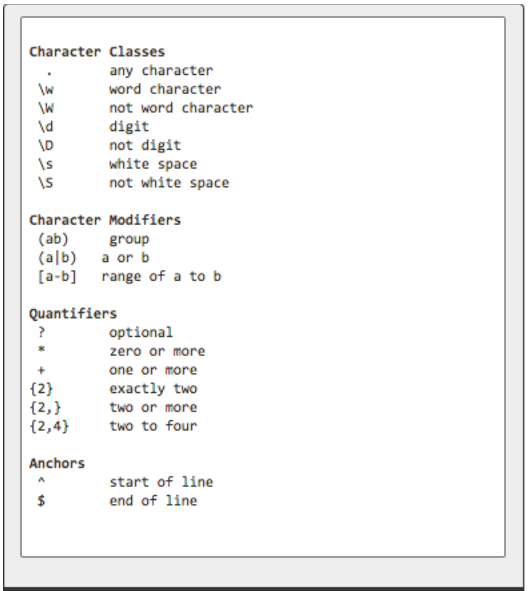

Figure 15: Regular Expressions Metacharacters

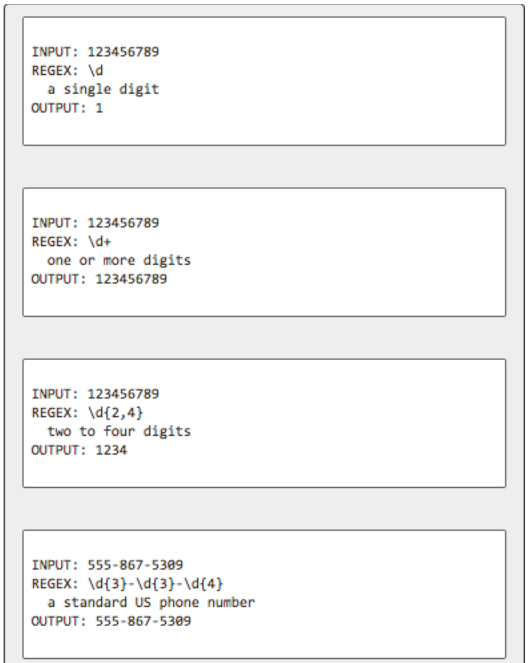

Figure 16: Example Regular Expressions

Server Variables are provided by Apache and are accessed using the following syntax: %{VARIABLE_NAME}. These variables can be used to access the request such as which HTTP method is being used or what IP is making the request. They can also be used to access system variables, internal server information, and more [55, 56] as illustrated in figure 17. Mod_rewrite also supports the ability to drill down in to additional HTTP, SSL, or environmental variables using the following syntaxes: %{HTTP:variable-name}, %{SSL:variable-name}, or %{ENV:variable-name}.

This ability to refine the request is utilized by the framework to determine if an AJAX request has been made. AJAX requests typically include a non-standard header in the form of X-Requested-With:XMLHttpsRequest, which can be detected using the following refined HTTP variable: %{HTTP:X-Requested-With}.

Flags are also provided by Apache to chain rules, write custom variables, and more [55, 56] as illustrated in figure 18.

```
HTTP Header variables
  HTTP_USER_AGENT
  HTTP_REFERER
  HTTP_COOKIE
  HTTP_FORWARDED
  HTTP_HOST
  HTTP_PROXY_CONNECTION
  HTTP_ACCEPT

Connection and Request Variables
  REMOTE_ADDR
  REMOTE_HOST
  REMOTE_USER
  REMOTE_IDENT
  REQUEST_METHOD
  SCRIPT_FILENAME
  PATH_INFO
  QUERY_STRING
  AUTH_TYPE

Server Internal Variables
  DOCUMENT_ROOT
  SERVER_ADMIN
  SERVER_NAME
  SERVER_ADDR
  SERVER_PORT
  SERVER_PROTOCOL
  SERVER_SOFTWARE

System Variables
  TIME_YEAR
  TIME_MON
  TIME_DAY
  TIME_HOUR
  TIME_MIN
  TIME_SEC
  TIME_WDAY
  TIME

mod_rewrite variables
  API_VERSION
  THE_REQUEST
  REQUEST_URI
  REQUEST_FILENAME
  IS_SUBREQ
```

Figure 17: mod_rewrite server variables

```
C       chain
CO      cookie=NAME:VALUE
E       env=NAME:VALUE
F       forbidden
G       gone
L       last
N       next
NC      nocase
QSA     qsappend
QSD     qsdiscard
R       redirect=CODE
S       skip=NUM

-d      existing directory
-f      existing file
```

Figure 18: mod_rewrite flags

Mod_rewrite is typically used for **clean URIs**—search engine optimized (SEO) friendly URIs that are purely structural and do not contain a query string, only the path of the resource. This is used for aesthetic and usability optimizations, as they are more human-readable. Clean URLs are also used to hide implementation information such as which server-side language is being used [56]. Most frameworks simply pass the entire path into a routing script using the rewrite script illustrated in figure 19.

```
RewriteEngine On

RewriteCond %{REQUEST_URI} !-f
RewriteCond %{REQUEST_URI} !-d
RewriteCond %{REQUEST_URI} !^index.php
RewriteRule ^(.*)$ index.php?q=$1 [QSA,L]
```

Figure 19: a typical rewrite script for Clean URLs

This script examines the request, if the request is not an existing folder or directory, indicated by the -f and -d flags, and is not the index.php file, then the request is passed to the index.php file for processing with the request appended as the value of the key q. The index.php file can then process the q key as desired.

Unfortunately, this approach does not allow for multiple clients as separate entities. It also allows malformed data to enter into the PHP script, which if not properly handled, may become a security vulnerability.

The proposed framework moves the processing of the URI from the index file to the .htaccess file and performs the following tasks:

- Routes requests to clients or API based on if the request is a XMLHttpsRequest
- Routes request to appropriate client cased on browser agent
- Re-formats requests to API or client resulting in a finalized form of: controller/action/id

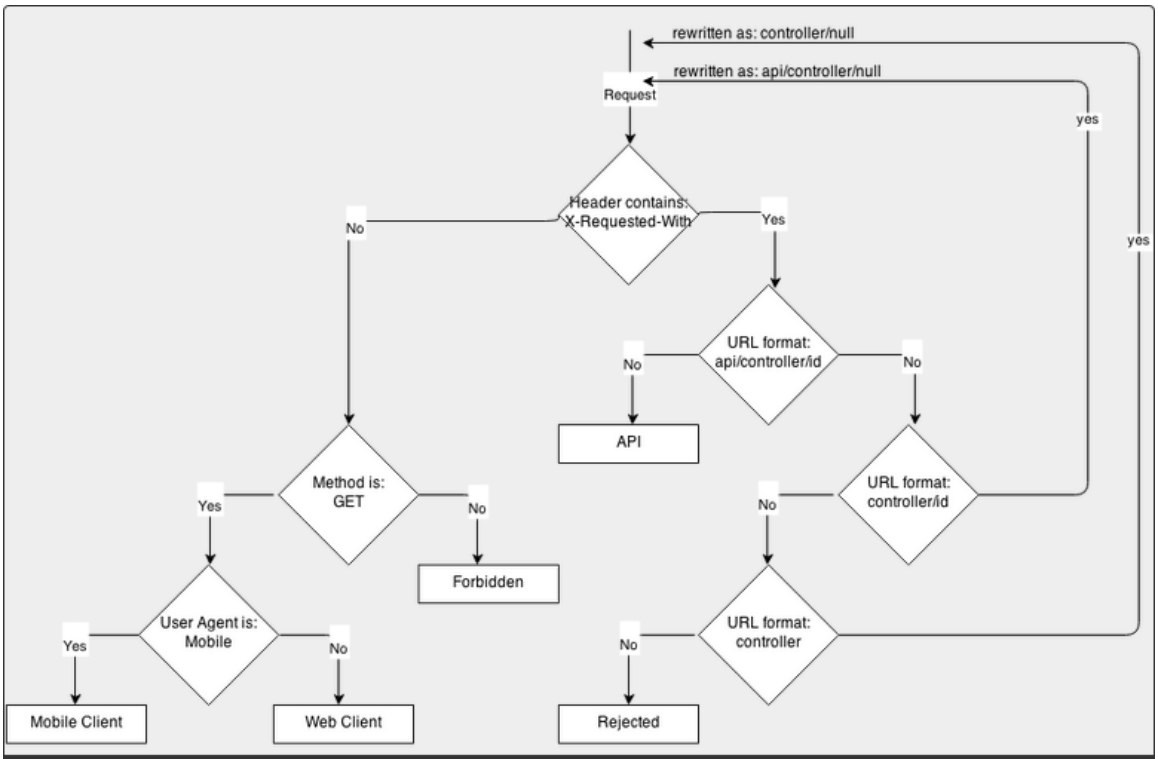

Figure 20: mod_rewrite flowchart

```
# Disallow Directory Listings
Options -Indexes

RewriteEngine On

# API
# Final
RewriteRule ^api/index.php/(\w+)/(\w+)/? api/index.php [L,NC,QSA,E=CONTROLLER:$1,E=ID:$2]

# API
# :controller:id
RewriteCond %{REQUEST_METHOD} ^(OPTIONS|PUT|DELETE|POST|GET)
RewriteCond %{HTTP:X-Requested-With} !^$
RewriteCond %{REQUEST_URI} !index.php
RewriteRule ^/?(.*)/(.*)/?$ api/index.php/$1/$2 [QSA,L]

# API
# :controller
RewriteCond %{REQUEST_METHOD} ^(OPTIONS|PUT|DELETE|POST|GET)
RewriteCond %{HTTP:X-Requested-With} !^$
RewriteCond %{REQUEST_URI} !index.php
RewriteRule ^/?(.*)/?$ api/index.php/$1/NULL [QSA,L]


# CORS preflight
RewriteCond %{REQUEST_METHOD} ^OPTIONS
RewriteRule ^.*$ api/index.php [L,E=CORS:TRUE]

# Block all Request Methods not directed to API
RewriteCond %{REQUEST_METHOD} ^(OPTIONS|PUT|DELETE|POST)
RewriteRule .* - [F]

# Mobile Client
RewriteCond %{HTTP_USER_AGENT} iphone|ipad|android|blackberry [NC]
RewriteCond %{REQUEST_METHOD} ^GET
RewriteCond %{HTTP:X-Requested-With} ^$
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ mobile/index.php?$1 [QSA,L]


# Web Client
RewriteCond %{REQUEST_METHOD} ^GET
RewriteCond %{HTTP:X-Requested-With} ^$
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ web/index.php?$1 [QSA,L]
```

Figure 21: proposed framework's rewrite script

This approach allows each client to function as separate entities, allows these entities to modify their own behavior, and prevents malformed data from accessing any server-side code all through the use of a few rules, conditions, and regular expressions. In addition, as an Apache module is performing this work, it is not limited to a single server-side language as is the current technique. In fact, this router can be used by any language Apache supports including PHP, Perl, Ruby, Python, and many more.

6.5 The API

The majority of functionality is performed by the API using a RESTful interface for client interactions. When a request is received, the API checks for a custom controller. If this controller does not exist, the API checks for a model of the same name. If the model exists, it can opt to utilize a generic controller. The generic controller is able to provide the standard REST interface for models that are only composed of data type components. More complicated models, or models that use child-objects of another class, such as comments, require a small amount of additional code to generate that parent-object. If no controller or model is found, a HTTP status of 404 is returned indicating the URI was not found.

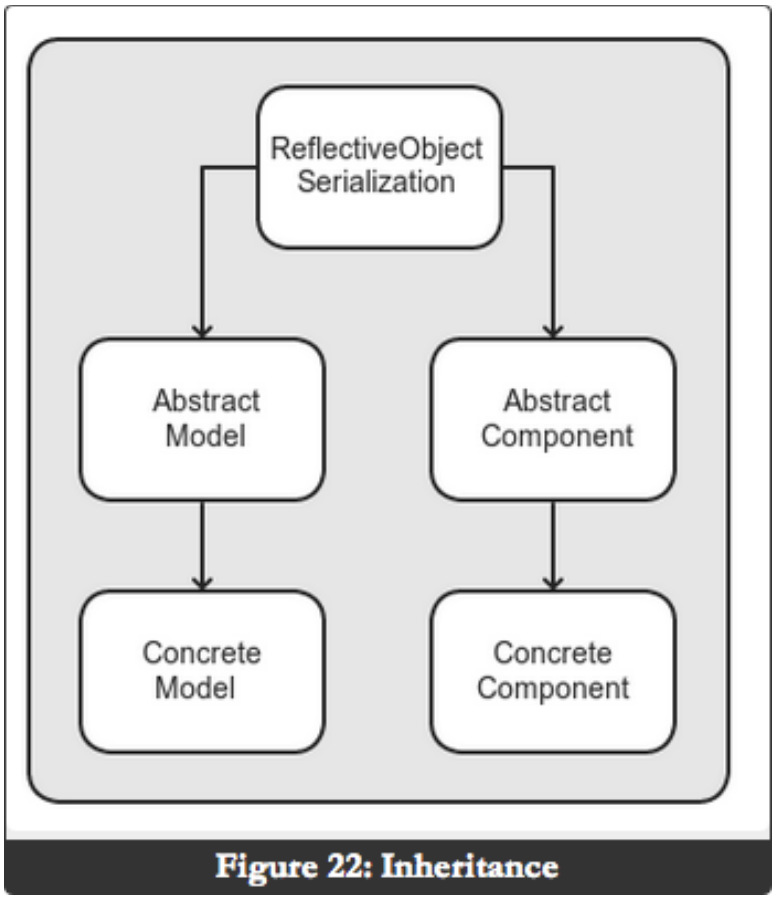

Figure 22: Inheritance

Once the controller is selected, it requests all models through a model factory. These model factories use a strategy pattern to allow for dynamically extended functionality at run time. Each

model factory is instantiated with a specific database driver that allows the framework to utilize any database it has a driver for including NoSQL databases. Multiple model factories may be instantiated utilizing multiple databases. This allows the programmer implementing the API to take advantage of each database's strengths where appropriate while still presenting a common interface to clients.

For this project, the MongoDB driver was used. MongoDB is the leading NoSQL database, and provides document-oriented storage, full indexing, replication and high availability, auto-sharding, in-place updates, ad-hoc querying, and built-in MapReduce support [58, 59].

After the model factory has been instantiated with a database driver, and the controller requests a model from the model factory, the model factory injects the database driver into the model, and then returns it to the controller.

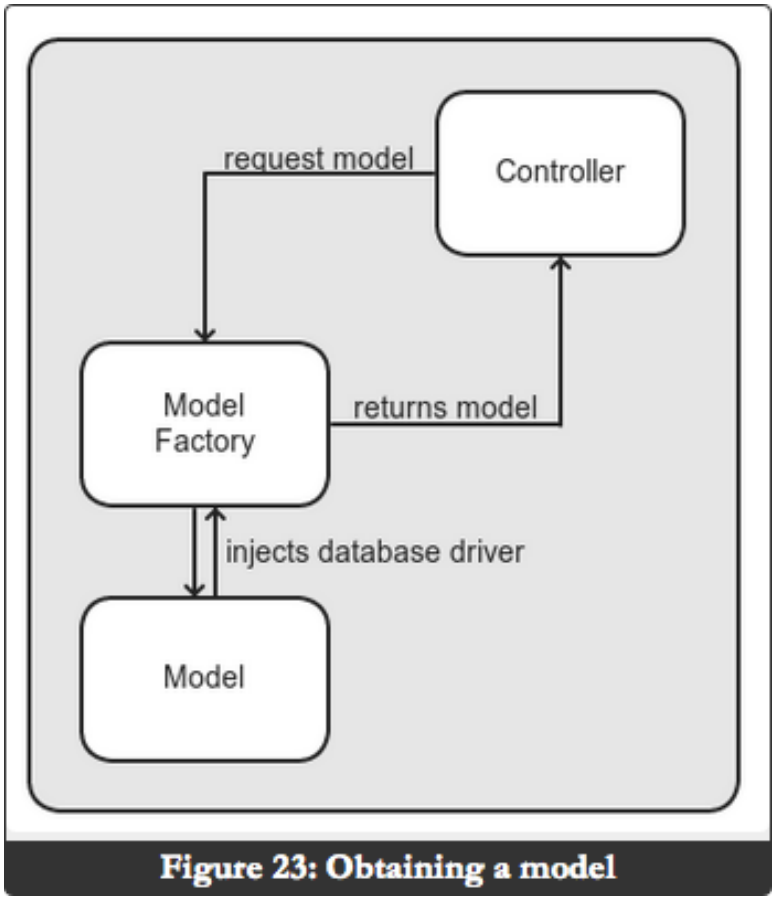

Figure 23: Obtaining a model

Once the model has been returned, the controller is able to fully interact with it accessing any available functionality that has been implemented. The model delegates property validation to the specified data type component and delegates database functionality such as creating and updating tables, and CRUD functionality to the database driver.

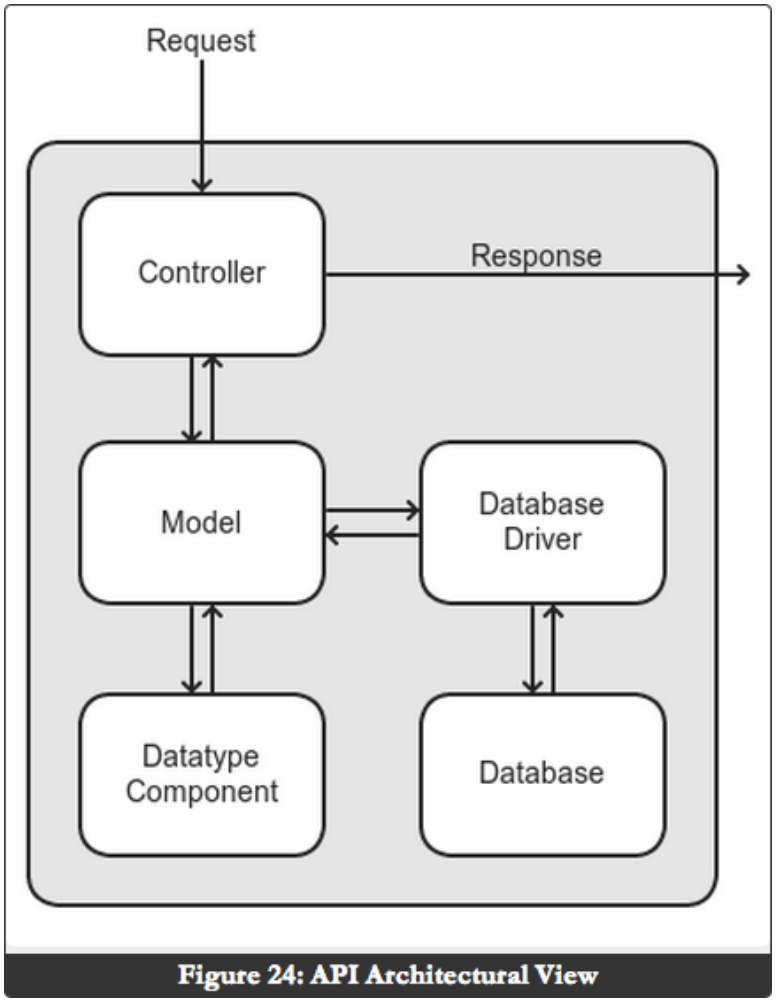

Figure 24: API Architectural View

The ReflectiveObject parent class provides basic model functionality such as non-array accessors and modifiers. This class uses reflection, the ability of a computer program to examine and modify its structure and behavior at runtime as well as the functionality for serializing models into JSON for easy processing by the database drivers [60, 61]. This parent class is also used by the controller as well as data type components.

Additional model-specific functionality such as id functionality, validation and CRUD operations are then added by the abstract Model subclass. All concrete models must inherit from this class and are constructed from data type components.

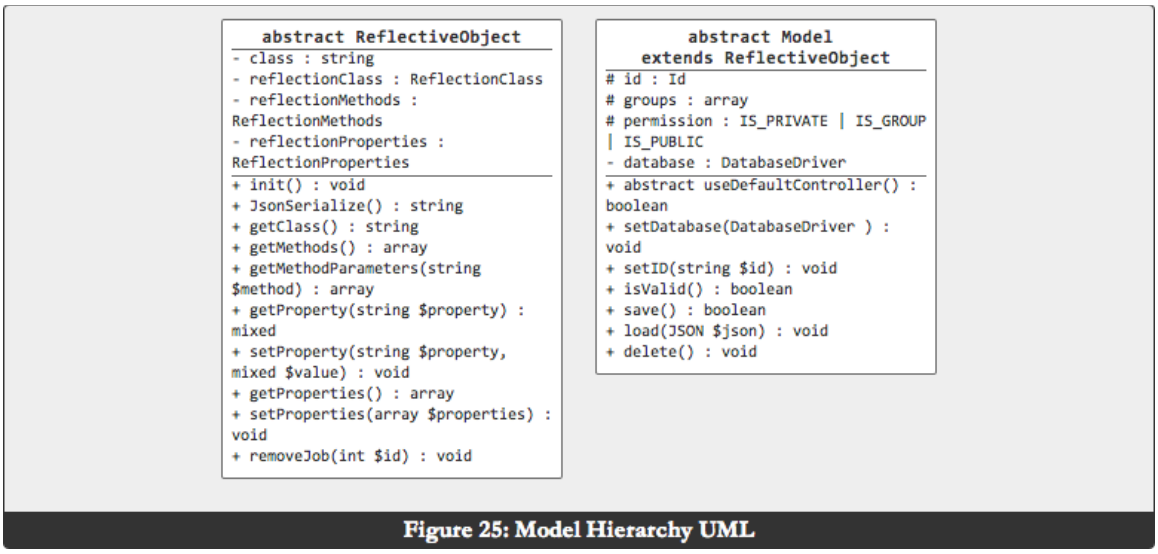

Figure 25: Model Hierarchy UML

Model creation is illustrated in figure 20. Public and protected properties will be serialized, private properties will not. The useDefaultController method returns a Boolean indicating if the default controller can be used. The construct method instantiates the appropriate model/data type components and sets the appropriate properties such as if it is required. Values for these components are set through the setProperty method inherited from ReflectiveObject, as is serialization.

For array values such as tags or comments, a small amount of custom code is required to verify the comment is valid, then add it it to the array. No additional code is required for standard functionality since everything else is already implemented through the model components, base Model class, or the ReflectiveObject class.

```
class Blog extends Model {

  protected $author;
  protected $title;
  protected $date;
  protected $tags;
  protected $post;
  protected $comments;


  public function useDefaultController() {
    return false;
  } // end function


  public function __construct() {
    $this->permission = Model::IS_GROUP;

    $this->author = new String();
    $this->author->setProperty('required', true);

    $this->title = new String();
    $this->title->setProperty('required', true);

    $this->date = new Date();
    $this->date->setProperty('required', true);

    $this->post = new RichText();
    $this->post->setProperty('required', true);

    $this->tags = array();
    $this->comments = array();
  } // end function


  public function addTag(Tag $tag) {
    if ($t->isValid())
      $this->tags[] = $t;
    else
      throw new Exception('Cannot add invalid tag: ' . $tag);

  } // end function


  public function addComment(Comment $comment) {
    if ($comment->isValid())
      $this->comments[] = $comment;
    else
      throw new Exception('Cannot add invalid comment: ' . $comment);
  } // end function
```

Figure 26: The Blog Model

6.5.1 Model/Data type Components

Data type components are reusable elements that modularize data validation and cleaning. Instead of writing duplicate code in every model that uses a data type such as date, data is created as a data type component and the model delegates all cleaning, normalization, and validation to the component. This prevents accidental errors in which code is not validated in all occurrences.

Since each data type is specified as a class, and this class information is passed to all clients, this also enables clients to automate a standardized input method for each class. These input methods maybe pop-up calendars, slides, drop-down lists, rich-text-areas, etc.

These components also inherit from ReflectiveObject allowing them to get and set properties and serialize to JSON.

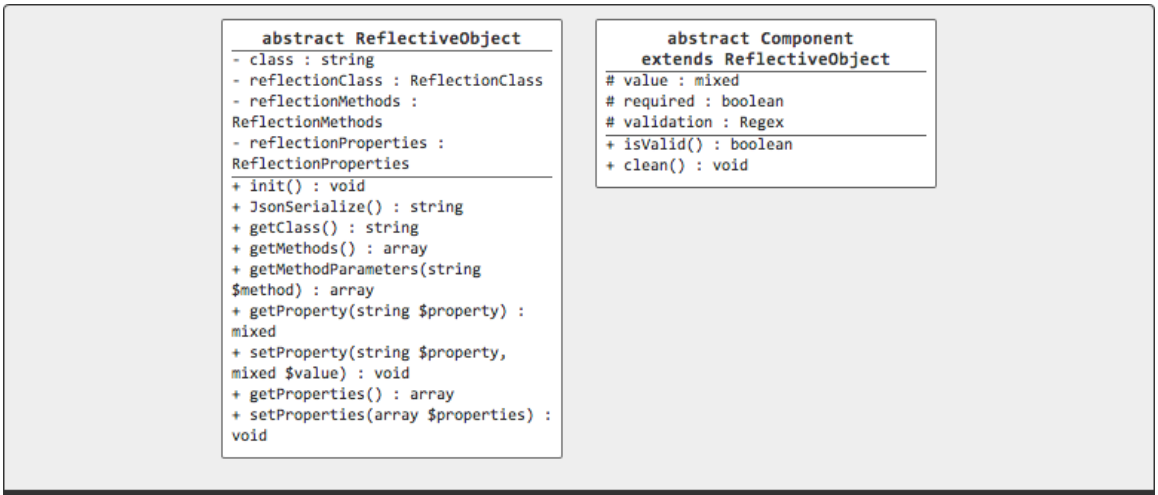

Figure 27: Core Classes UML

6.5.2 Database Drivers

Database drivers can be established for any existing database. They must implement the DBDriver interface that includes methods to construct, find, save, and delete models passed in as JSON. Drivers should include functionality to automatically create and update tables and gracefully handle missing information. At this time, MongoDB is the only database driver provided.

6.6 Clients

Each client is responsible for the functionality and presentation required by its specific implementation. The majority of functionality will be provided by the API; however, in special cases such as mobile, additional functionality maybe required for local caching or changing IP addresses.

These clients may using custom code, rely on server-side languages, or utilize one of the many client-side JavaScript libraries such as AngularJS, BackBone.js, Ember.js, Knockout, etc. or the JavaScript library for clients provided by this project. This combination provides as much functionality as possible, while also allowing the programmer implementing the interface as much freedom as possible.

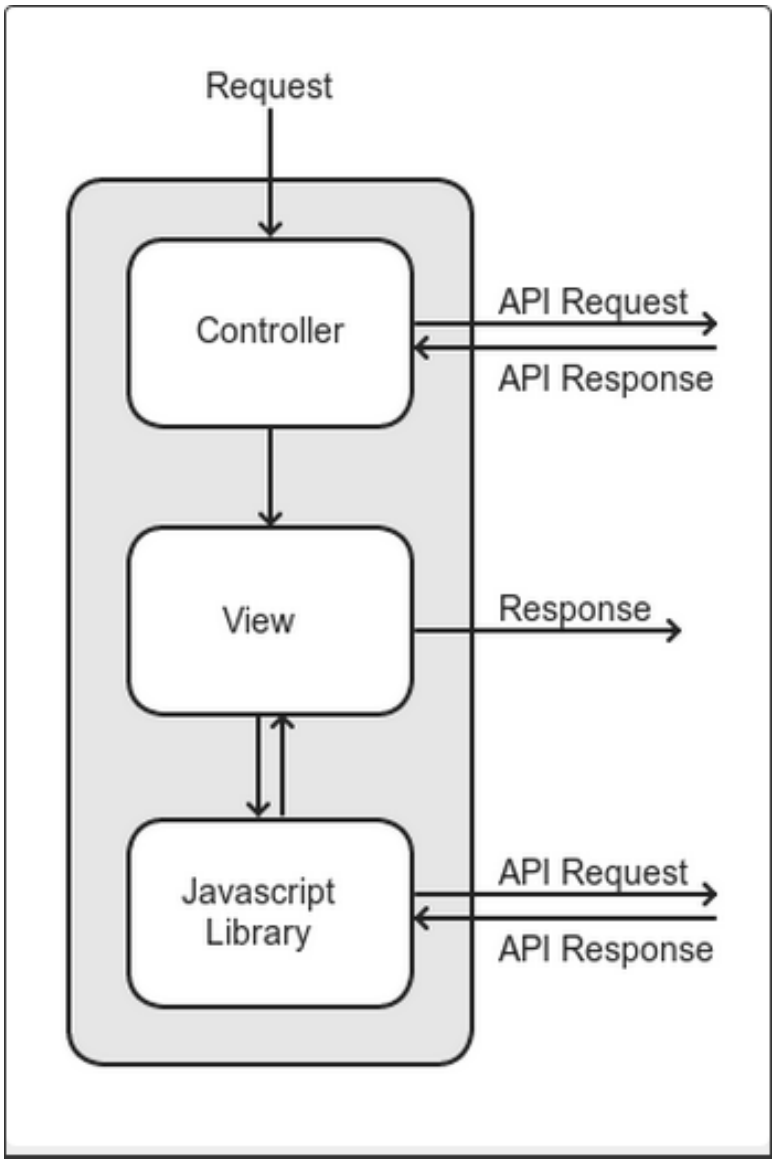

Figure 28: Client Architectural View

6.7 JavaScript Library for Clients

For maximum efficiency, a small JavaScript library is also provided to clients to help generate standard views such as the lists view, new/edit view, and detail view utilizing XHR requests to avoid unnecessary page refreshes [22] and JavaScript promises to chain multiple events. This Promise interface represents a proxy for a value not necessarily known when the promise is created. It allows a programmer to associate handlers to an asynchronous action's eventual success or failure which allows asynchronous methods to return values like synchronous methods: instead of the final value, the asynchronous method returns a promise of having a value at some point in the future [67, 68].

These views are generated by using the HTTP OPTIONS method, which returns information on each model data type component including name, value, type, and validation. This information is then used to create the appropriate HTML elements such as forms and fields tagged with the appropriate data attributes and CSS classes, as well as validation events which will indicate erroneous input in real-time.

All indicators are done through the addition and deletion of CSS classes, as it is a bad practice to programmatically alter the HTML elements [71]. This allows the programmer implementing the interface to determine how errors are visually indicated and provides a hook for additional extended JavaScript functionality as well as data attributes that can be selected and styled with CSS.

To allow external servers access to the framework's API, this project implements the cross-origin resource sharing (CORS) mechanism—a W3C recommendation to provide a way for web servers to support cross-site access which has been previously restricted by modern browsers due to security concerns. This allows the project to provide cross-site access controls and enable secure cross-site data transfers making a user's data available at any site [69, 70].

7 CONCLUSION

This project developed an extensible, rapid-development framework with minimal learning curve that satisfies all project goals.

Desktop, mobile, and API clients support are provided using Apache's mod_rewrite for routing by examining incoming requests for http methods, custom headers, and user agents. This provides an API-centric foundation allowing for easy extensibility to additional clients and devices.

Automatic database generation and basic create, read, update, and delete functionality is implemented by utilizing reflective models and intelligent, interchangeable database drivers able to update database schemas as required. This eliminated duplicate code between the server and database tiers preventing additional opportunities for coding errors.

Automatically generated, self-validating forms for list, edit, and detail view at the client level are accomplished using a JavaScript library utilizing XMLHttpRequest, JavaScript promises, cross origin scripting, regular expressions for validation, and the addition of the http OPTIONS method to the standard RESTful interface. This allows for rapid development across multiple clients by eliminating the coding of lists and forms this functionality would typically require. Further, as forms are dynamically created based on current state, it eliminates the need to update clients when API functionality has been modified. Finally, by establishing a short list of easy to understand JavaScript methods as well as data attributes and the appropriate CSS classes, front-end developers are able to easily retrieve pre-formatted data, and simply style it according to the needs of their particular client.

Data type components such as text, date, email, and phone numbers provide a consistent implementation of sanitation and validation preventing many common exploits includes XSS and SQL-injection attacks. This provides security by default and prevents the duplication of code

required to sanitize and validate code by migrating the process to the base components instead of at each occurrence.

All of this functionality is provided by an efficient, secure, extensible framework that is easy to learn. Simply create a declarative model with the necessary fields and the database, API, and basic client side interface are all automatically generated and ready to be styled. All changes to the model are automatically reflected at all levels with no further coding is required.

8 FUTURE WORK

Although this project satisfied all project goals, it would benefit from additional functionality, refinements, and testing including usage in real-world applications, additional research to resolve limitations in current routing, the development of additional database drivers, and opening the API to third-party clients.

To enable testing in real world applications, this framework will be released as open source on Github.com and at www.CandygramPHP.com accompanied by an introduction to the framework, documentation, and tutorials.

The current approach using mod_rewrite requires additional research as it removes POST and PUT data when forwarding requests. Currently, this issue has been resolved by serializing form data into the query string appended to the URI which is forwarded. However, this is subject to limitations and not satisfactory as a long-term solution.

Although MongoDB is the most popular NoSQL database, SQL databases are even more common which requires the development of additional database drivers for use in the majority of development environments. It would also appear RESTful APIs are more consistent with the relational data model instead of direct object stores.

To extend this framework to third-party clients, the OAuth 2.0 standard needs to be implemented. OAuth 2.0 is an open standard for authorization that provides client applications a secure delegated access to server resources on behalf of a resource owner; in other words, OAuth provides a method for users to use third party clients using an authorization server without worrying about their access credentials such as username and password being compromised [64, 65, 66].

9 REFERENCES

[1] Wikipedia contributors, "Multitier Architecture" [Online]. Available: http://en.wikipedia.org/wiki/Multitier_architecture. [Accessed 7 August 2014]

[2] Wikipedia contributors, "Object-oriented programming" [Online]. Available: http://en.wikipedia.org/wiki/Object-oriented_programming. [Accessed 7 August 2014]

[3] Wikipedia contributors, "Prototype-based programming" [Online]. Available: http://en.wikipedia.org/wiki/Prototype-based_programming. [Accessed 7 August 2014]

[4] Wikipedia contributors, "Relational Model" [Online]. Available: http://en.wikipedia.org/wiki/Relational_model. [Accessed 7 August 2014]

[5] Ecma International, "ECMAScript Language Specification" [Online]. Available: http://www.ecma-international.org/ecma-262/5.1/. [Accessed 7 August 2014]

[6] Network Working Group, "Hypertext Transfer Protocol" [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html. [Accessed 7 August 2014]

[7] Wikipedia contributors, "Object-relational mapping" [Online]. Available: http://en.wikipedia.org/wiki/Object-relational_mapping. [Accessed 7 August 2014]

[8] U.S. Department of Commerce, "Computer and Internet Use in the United States" [Online]. Available: http://www.census.gov/prod/2013pubs/p20-569.pdf. [Accessed 7 August 2014]

[9] V. Maxus, "The Growth Of The Internet Over The Past 10 Years â€“ Infographic" [Online]. Available: http://www.themainstreetanalyst.com/2012/08/22/the-growth-of-the-internet-over-the-past-10-years-infographic/. [Accessed 7 August 2014]

[10] Susannah Fox and Lee Rainie, "The Web at 25 in the U.S.", 27 February 2014 [Online]. Available: http://www.pewinternet.org/2014/02/27/the-web-at-25-in-the-u-s/. [Accessed 7 August 2014]

[11] Symantec Corporation, "Internet Security Threat Report 2014", April 2014 [Online]. Available: http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf. [Accessed 7 August 2014]

[12] David Gitonga, "A Timeline of Companies That Have Been Hacked In 2013", 12 March 2013 [Online]. Available:http://heavy.com/tech/2013/03/a-timeline-of-companies-that-have-been-hacked-in-2013/. [Accessed 7 August 2014]

[13] Information is Beautiful, "World's Biggest Data Breaches", 2014 [Online]. Available:http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/. [Accessed 7 August 2014]

[14] Network Working Group, "Uniform Resource Identifier (URI): Generic Syntax", 2005 [Online]. Available:http://www.ietf.org/rfc/rfc3986.txt. [Accessed 8 August 2014]

[15] W3C, "Indexed Database API", 4 July 2013 [Online]. Available:http://www.w3.org/TR/IndexedDB/. [Accessed 8 August 2014]

[16] W3C, "Semantic Web", 2013 [Online]. Available:http://www.w3.org/standards/semanticweb/. [Accessed 8 August 2014]

[17] Wikipedia contributors, "Semantic Web" [Online]. Available: http://en.wikipedia.org/wiki/Semantic_Web. [Accessed 8 August 2014]

[18] Mozilla Developer Network, "HTML5 element list" [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5/HTML5_element_list. [Accessed 8 August 2014]

[19] W3C, "CSS Specifications" [Online]. Available: http://www.w3.org/Style/CSS/specs.en.html. [Accessed 8 August 2014]

[20] Less Core Team, "Language Features" [Online]. Available: http://lesscss.org/features/. [Accessed 8 August 2014]

[21] Hampton Catlin, Natalie Weizenbaum, Chris Eppstein, and numerous contributors., "CSS with superpowers" [Online]. Available: http://sass-lang.com/. [Accessed 8 August 2014]

[22] Mozilla Developer Network, "XMLHttpRequest" [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest. [Accessed 8 August 2014]

[23] The jQuery Foundation, "jQuery: write less, do more." [Online]. Available: http://jquery.com/. [Accessed 8 August 2014]

[24] Pty Ltd, "jQuery Usage Statistics" [Online]. Available: http://trends.builtwith.com/javascript/jQuery. [Accessed 8 August 2014]

[25] Wikipedia Contributors, "Content Delivery Network" [Online]. Available: http://en.wikipedia.org/wiki/Content_delivery_network. [Accessed 8 August 2014]

[26] W3C, "Extensible Markup Language (XML)" [Online]. Available: http://www.w3.org/XML/. [Accessed 8 August 2014]

[27] ECMA International, "The JSON Data Interchange Format" [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. [Accessed 8 August 2014]

[28] W3C, "Introduction to Web Components" [Online]. Available: http://www.w3.org/TR/components-intro/. [Accessed 8 August 2014]

[29] Wikipedia Contributors, "Polyfill" [Online]. Available: http://en.wikipedia.org/wiki/Polyfill. [Accessed 8 August 2014]

[30] Eric Freeman and Elisabeth Freeman, "Head First Design Patterns". Sebastopol, CA: O'Reilly Media, Inc, 2004, ch.12, pp. 500-549.

[31] Eric Redmond and Jim R. Wilson, "Seven Databases in Seven Weeks". Dallas, TX: Pragmatic Programmers, LLC., 2012.

[32] Micahael Howard, David LeBlanc and John Viega, "24 Deadly Sins of Software Security". New York, NY: McGraw Hill, 2010.

[33] OWASP Contributors, "Top 10 1013 - Top 10" [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-Top_10. [Accessed 8 August 2014]

[34] OWASP Contributors, "SQL Injection" [Online]. Available: https://www.owasp.org/index.php/SQL_Injection. [Accessed 8 August 2014]

[35] OWASP Contributors, "Cross-site Scripting (XSS)" [Online]. Available: https://www.owasp.org/index.php/XSS. [Accessed 8 August 2014]

[36] Wikipedia Contributors, "Web application framework" [Online]. Available: http://en.wikipedia.org/wiki/Web_application_framework. [Accessed 8 August 2014]

[37] Bootstrap Core Team, "Getting Started" [Online]. Available: http://getbootstrap.com/getting-started/. [Accessed 8 August 2014]

[38] Pty Ltd, "Twitter Bootstrap Usage Statistics" [Online]. Available: http://trends.builtwith.com/docinfo/Twitter-Bootstrap. [Accessed 8 August 2014]

[39] Google, "HTML enhanced for web apps!" [Online]. Available: https://angularjs.org/. [Accessed 8 August 2014]

[40] Pty Ltd, "Angular JS Usage Statistics" [Online]. Available: http://trends.builtwith.com/javascript/Angular-JS. [Accessed 8 August 2014]

[41] WordPress Core Team, "About WordPress" [Online]. Available: http://wordpress.org/about/. [Accessed 8 August 2014]

[42] Pty Ltd, "WordPress Usage Statistics" [Online]. Available: http://trends.builtwith.com/cms/WordPress. [Accessed 8 August 2014]

[43] Robert Westervelt, "Popular WordPress E-Commerce Plugins Riddled With Security Flaws" 18 June 2013 [Online]. Available: http://www.crn.com/news/security/240156883/popular-wordpress-e-commerce-plugins-riddled-with-security-flaws.htm. [Accessed 8 August 2014]

[44] David Heinemeier Hansson, "Lean all about Ruby on Rails" [Online]. Available: http://rubyonrails.org/documentation/. [Accessed 8 August 2014]

[45] Pty Ltd, "Ruby on Rails Usage Statistics" [Online]. Available: http://trends.builtwith.com/framework/Ruby-on-Rails. [Accessed 8 August 2014]

[46] Lynn Beighley and Mchael Morrison, "Head First PHP & MySQL". Sebastopol, CA: O'Reilly Media, Inc, 2008.

[47] Ben Mills, "How to Create an Object-Oriented Blog Using PHP" [Online]. Available: http://code.tutsplus.com/tutorials/how-to-create-an-object-oriented-blog-using-php--net-1230. [Accessed 8 August 2014]

[48] Matt Griffin, "Client Relationships and the Multi-Device Web" 23 July 2013 [Online]. Available: http://alistapart.com/article/client-relationships-and-the-multi-device-web. [Accessed 8 August 2014]

[49] W3C, "SOAP Version 1.2 Part 0: Primer" [Online]. Available: http://www.w3.org/TR/2007/REC-soap12-part0-20070427/. [Accessed 8 August 2014]

[50] Wikipedia Contributors, "SOAP" [Online]. Available: http://en.wikipedia.org/wiki/SOAP. [Accessed 8 August 2014]

[51] Wikipedia Contributors, "Representational state transfer" [Online]. Available: http://en.wikipedia.org/wiki/Representational_state_transfer. [Accessed 8 August 2014]

[52] Ludovico Fischer, "A Beginner's Guide to HTTP and REST" [Online]. Available: http://code.tutsplus.com/tutorials/a-beginners-guide-to-http-and-rest--net-16340. [Accessed 8 August 2014

][53] Bitworking, "How to create a REST Protocol" [Online]. Available: http://bitworking.org/news/How_to_create_a_REST_Protocol. [Accessed 8 August 2014

][54] Network Working Group, "Hypertext Transfer Protocol" [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html. [Accessed 8 August 2014]

[55] The Apache Software Foundation, "Apache Module mod_rewrite" [Online]. Available: http://httpd.apache.org/docs/current/mod/mod_rewrite.html. [Accessed 8 August 2014]

[56] Rich Bowen, "The Definitive Guide to Apache mod_rewrite". Berkeley, CA: Apress, 2006.

[57] Wikipedia Contributors, "Regular expression" [Online]. Available: http://en.wikipedia.org/wiki/Regular_expression. [Accessed 8 August 2014]

[58] Kristina Chodorow, "MongoDB: the definitive guide," 2nd ed. Sebasopol, CA: O'Reilly Media, Inc, 2013.

[59] MongoDB, Inc., "The MongoDB 2.6 Manual" [Online]. Available: http://docs.mongodb.org/manual/. [Accessed 8 August 2014]

[60] Wikipedia Contributors, "Reflection (computer programming)" [Online]. Available: http://en.wikipedia.org/wiki/Reflection_(computer_programming). [Accessed 8 August 2014]

[61] The PHP Group, "Reflection " [Online]. Available: http://php.net/manual/en/book.reflection.php#book.reflection. [Accessed 8 August 2014]

[62] The Apache Software Foundation, "Authentication and Authorization" [Online]. Available: http://httpd.apache.org/docs/2.2/howto/auth.html. [Accessed 8 August 2014]

[63] Wikipedia Contributors, "Standard Generalized Markup Language" [Online]. Available: http://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language. [Accessed 8 August 2014]

[64] Internet Engineering Task Force, "The OAuth 2.0 Authorization Framework" [Online]. Available: http://tools.ietf.org/html/rfc6749. [Accessed 8 August 2014]

[65] Eran Hammer-Lahav, "Explaining OAuth" [Online]. Available: http://hueniverse.com/2007/09/05/explaining-oauth/. [Accessed 8 August 2014]

[66] Eran Hammer-Lahav, "The OAuth 1.0 Guide" [Online]. Available: http://hueniverse.com/oauth/guide/. [Accessed 8 August 2014]

[67] Mozilla Developer Network, "HTTP access control (CORS)" [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS. [Accessed 8 August 2014]

[68] Monsur Hossain, "Using CORS" 29 October 2013 [Online]. Available: http://www.html5rocks.com/en/tutorials/cors/. [Accessed 8 August 2014]

[69] Mozilla Developer Network, "Promise" [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise. [Accessed 8 August 2014]

[70] Jake Archibald, "JavaScript Promises: there and back again" 29 January 2014 [Online]. Available: http://www.html5rocks.com/en/tutorials/es6/promises/. [Accessed 8 August 2014]

[71] W3C, "JavaScript best practices" [Online]. Available: http://www.w3.org/wiki/JavaScript_best_practices. [Accessed 8 August 2014]