The Non-Traditional Programs Scheduling Assistant:

A Web Application for Determining Course Availability

A Capstone Report submitted to the
Regents Bachelor of Arts Program at
WVUP-Parkersburg

In Partial Fulfillment of
the requirements for the degree of
Regents Bachelor of Arts
Computer Science

by

Thomas Byrne

Jenny Dawkins, Professor
Robin Ambrozy
Landon Cole

ABSTRACT

Currently a student may need to use up to three different, awkward systems in order to determine which classes to sign up for. This burden is often put from the student onto their advisor, especially in the Non-Traditional Programs, where students may not be familiar or comfortable with the complicated systems currently offered by WVUP. In order to reduce staff involvement and streamline this process, a new web-based application was developed. This application provides a single, unified view into all the courses offered by WVUP, as well as all the degree requirements that they fulfill.

This application allows the Non-Traditional Programs staff to quickly and easily answer incoming student requests, as well as enabling students to avoid burdening their advisors or NTP staff.

The objectives of this application are:

1. Provide a mechanism to codify and coalesce all the domain knowledge about which degree requirements each WVUP course fulfills.
2. Provide a system that contains all the pertinent information from WVUP's course schedule, course catalog and other systems into a single knowledge base.
3. Provide a simple user interface that allows a user to search this knowledge base for relevant courses for each user's requirements.

Table Of Contents

ABSTRACT.....ii

LIST OF FIGURES & ILLUSTRATIONS.....v

LIST OF TABLES.....vi

SECTION

1 - INTRODUCTION.....1

1.1 Project Introduction & Background.....1

DEFINING A SOLUTION.....3

2.1 Solution Overview.....3

2.2 Success Metrics.....3

3. SOLUTION REQUIREMENTS.....4

3.1 End User Requirements.....4

3.2 Software Requirements.....4

3.2.1 Server Software Requirements.....4

3.2.1 End User Software Requirements.....4

3.3 Hardware Requirements.....4

3.4 Constraints.....5

3.4.1 Internal System Constraints.....5

3.4.2 Cost Constraints.....5

TECHNOLOGY EVALUATION AND CHOICES.....6

4.1 Database.....7

4.2 Application Server.....8

4.3 User Interface.....9

4.4 Monitor Application.....10

5. ARCHITECTURE.....11

5.1 Database Architecture.....11

5.2 Server.....11

5.2.1 Data Access Layer.....11

5.2.3 HTTP Handling Architecture.....13

5.3 User Interface Architecture.....15

5.3.1 Login Page.....15

5.3.2 Search Page.....15

5.3.3 Administration Page.....15

5.4 Build and Deployment.....16

DATABASE DEFINITIONS AND DATA STRUCTURES.....18

6.1 Setup.....18

6.2 Course Metadata.....18

6.3 Course Instances 18
APPLICATION SERVER FUNCTIONS..19
7.1 HTTP/Web Server 19
7.2 Authentication & Authorization 19
7.3 Database Access 20
7.3.1 Database Setup 21
7.4 Information..21
7.5 Scheduling and Executing Background Jobs 21
7.6 Data Import from WVUP Systems 21
8. MONITOR APPLICATION FUNCTIONS 23
8.1 Monitor NTPAppServer Process 23
8.2 Update NTPAppServer..23
8.3 Generate Manifest 24
9. USER INTERFACE OVERVIEW..25
9.1 LOGIN PAGE 25
9.2 STUDENT CLASS SEARCH PAGE 26
9.3 ADMINISTRATOR UPDATE PAGE 27
SUMMARY AND RESULTS 28
10.1 NTP Office 28
10.2 The Monitoring Application..28
Appendix A - REST Interface Objects & Actions..29
A.1 Example Requests 29
A.1.1 Info Request 29
A.1.2 Authentication Requests..30
A.1.2.1 - Validation Request/Response..30
A.1.2.2 - Invalidation Request/Response..31
A.1.3 Sample Search..31
Appendix B - Sample Manifest for Web Update. 35
Appendix C - List of External Libraries Used 36
Appendix D - Database Properties..37
APPENDIX E - DATABASE TABLES..38

LIST OF FIGURES

Figure 5.1 - Data Storage Interface Definition 11
Figure 5.2 - UML Class Diagram of Data Access 12
Figure 5.3 - Sequence Diagram of HTTP POST Method 14
Figure 9.1 - Login Page UI 24
Figure 9.2 - Search Page UI 25
Figure 9.3 - Administration Page UI 26

LIST OF TABLES

Table 4.1 - Database Technology Evaluation 7
Table 4.2 - Application and Web Server Technology Evaluation 8
Table 4.3 - User Interface Technology Evaluation 9
Table A.1 - Object and Action List 29
Table C.1 - List of External Libraries 36
Table D.1 - Database Properties 37
Table E.1 - Course Metadata DDL 38
Table E.2 - Course Instance DDL 39

1 - INTRODUCTION

1.1 Project Introduction & Background

The current workflow for scheduling which courses are available and fulfill specific degree requirements is deeply flawed. The information is available to students, but is often difficult to find and requires multiple systems and documents to determine each one. Currently the information that is needed resides in as many as three different locations and systems. The first is the university's published course schedule. This is a web page which is published automatically, and conveys which courses are being offered in each semester. There are different pages that must be navigated for each physical location (such as the Main Campus, or Jackson Center). For each location, there is a single page showing the total list of courses that is offered for an entire semester, and this list can easily be hundreds of courses long. In order to find out a comprehensive list of which courses are offered by the university, a student must visit WVUP's course catalog page, which is actually a 240 page long PDF document, showing every course that the university could possibly offer, without detailing which requirements they fulfill, or when they are offered, if at all. A third location offers Uniform Course Syllabi, which can be used to determine which requirements are fulfilled by each course, but again, there is no reference as to when or if the course is offered.

This leads to several problems and confusion when a student needs specific courses to fulfill their degree requirements. As an example, if a student is required to take 6 hours of humanities, and desires to take a literature class, he must first see what literature courses the college offers, then check the syllabus to ensure that they are considered humanities courses, and then find out when (or if) the course is offered in the upcoming semester. A student is required to use multiple systems, read or search multiple large documents repeatedly, just to find a small set of classes.

This process is awkward at best, or completely broken at worst, and based on the discussions with Non-Traditional Programs(NTP) staff, many students find it incomprehensible enough that they prefer just to have their advisor do it instead. This leads to the staff having to spend valuable hours performing tasks that the students should (and could) be doing. A new web system that distills all the needed information and makes it simple to search based on semester and requirements will allow the NTP staff to streamline the task when they are required to do it, and simplify the workflow enough that students can more easily do it themselves.

1.2 PROBLEM DEFINITION

The user interaction required to perform these necessary tasks is needlessly complicated, to the point where many students defer the task to their advisors. The advisors and staff members have contextual information that they have learned, and are able to navigate the systems to produce the needed results. The necessary information from all the systems and staff need to be put into a single system that is simple enough for students to use, but also useful and fast enough that the staff members will use it when required.

2. DEFINING A SOLUTION

2.1 Solution Overview

The determination was that a streamlined, targeted, search-based web application would be ideal to solve the process issues. This application would allow the student or staff to select a simple set of metrics (such as “Upper Division”, “Humanities”, and “Fall, 2013” and the system would retrieve a list of all courses that meet all of those requirements. This distills the systems required to perform this task from 3 to 1.

2.2 Success Metrics

The primary metric of success will be deploying a fully functional system, usable by the NTP staff to streamline their operations. Additional stretch goals include:

1) Adoption by students in the NTP program.

2) Integration into the NTP or WVUP web sites.

3. SOLUTION REQUIREMENTS

3.1 End User Requirements

The end-user requirements are as follows:

1. No technical knowledge is needed for setup or use.
2. Server and UI must be self-updating. No user interaction will be required for installation of new versions.
3. Contextual setup and initial data configuration by end-users should be minimal.
4. Search tools should be as self explanatory as possible.

3.2 Software Requirements

3.2.1 Server Software Requirements

- OS: Windows Vista or better, OR MacOSX 10.7 or better, OR Linux variant
- postgreSQL Server 9.2.3 or better (untested on pre-9.2.3 servers)
- Java version 7 or better

3.2.1 End User Software Requirements

- Any OS running a modern browser: Firefox 17 or better, IE 8 or better, Safari or Chrome

3.3 Hardware Requirements

The hardware requirements are defined only by the ability to run the appropriate software. The Application server requires 512 megabytes of memory allocation, and the Monitor app requires 256 megs. Other requirements can be found below.

- PostgreSQL requirements: http://www.postgresql.org/docs/9.0/static/supported-platforms.html
- Java - http://www.java.com/en/download/help/sysreq.xml

3.4 Constraints

3.4.1 Internal System Constraints

The NTP System is not allowed to access WVUP's internal systems directly. Since the WVUP Banner system is the system of record for courses and schedules, we need to get this data somehow. The only access we can have to this data is publicly available copies of the data. WVUP makes this data available via a published HTML page on the WVUP web site, so we will use that as our canonical data source.

3.4.2 Cost Constraints

The NTP program currently has no money in its budget to contribute for hardware or software. All software (development and deployment) must be free. The department has several computers that are adequate for running the server processes.

4. TECHNOLOGY EVALUATION AND CHOICES

For this project, a set of technologies needed to be chosen. For each part of the system, we made a determination as to an appropriate technology for that portion of the system. Two relevant choices that were made without evaluation were:

a) That the system be deployed as a Web application

b) That the system be primarily written in Java.

The logic for these choices were as follows: A Web application requires no end-user installations, nor are there any requirements on the end user other than the de-facto standard of having a relatively modern browser installed. The choice of Java as a primary language was primarily due to three factors: it is a mostly cross-platform language, it can be made to scale to any size of application, and lastly, I am familiar with it, thus removing the need to ramp up on at least one of the chosen technologies.

For each of the other technology choices, 6 factors were considered, plus additional factors, depending on the technology. These six primary factors are:

1. Is it an established technology?

   Has it been around long enough to have proven uses, and has it been adopted by other projects?

2. Is it a stable technology?

   Are there updates, new releases and bug fixes published? Are there constant issues surrounding the technology with regards to security?

3. Is it free?

The NTP program has no budget to invest in this project, so all software must be free.

4. Is it Open-Source Software(OSS)?

   While not a make-or-break factor, we will prefer OSS due to the ability to provide our own bug fixes and the ability to audit the code, as necessary.

5. Is it Cross-Platform?

   The software will be developed and run on multiple platforms. WVUP currently uses mostly Microsoft-based servers and workstations, while I use a MacOSX/ Darwin platform. Most “real world” servers are deployed on some variant of Linux or Unix.

6. How familiar am I with this technology?

   Due to the tight time constraints of this project, I will probably prefer technologies that I have some familiarity with.

4.1 Database

Table 4.1 - Database Technology Evaluation

Technology	Established/Stable	Free/OSS	Familiarity
BerkeleyDB	**Yes/Yes**	**Yes/Yes**	**High**
MySQL	**Yes/Yes**	**Yes/Yes**	**Medium**
postgreSQL	**Yes/Yes**	**Yes/Yes**	**Low**

All of the database choices that were evaluated had similar characteristics: they are all open source, have good documentation, have stable builds and active communities, and all three have the option of companies that provide service for hire. postgreSQL had the most positive information and details on high-volume transactions, as well as lots of information about setting up clusters and high-availability scenarios. I felt that if we wanted to achieve the stretch target of all of WVUP potentially using this system, that postgreSQL server was the best choice. In hindsight, based upon the simple schema and small amount of data (only a few thousand rows per year), any of the three could have been made to work. Even with scaling to an enterprise the size of WVUP with 4000-5000 students using this system, the throughput could easily be managed by one server and one database.

4.2 Application Server

Table 4.2 - Application and Web Server Technology Evaluation

Technology	Established/Stable	Free/OSS	Familiarity
Apache Tomcat	Yes/Yes	Yes/Yes	High
JBoss Netty	Yes/Yes	Yes/Yes	High
JBoss RESTEasy	Yes/Yes	Yes/Yes	None

The three candidates for the application server framework that were considered all had very similar scores, are relatively established and stable, and all are free and OSS. I took a close look at JBoss RESTEasy, but decided that having to learn the API's of a new framework ruled it out, due to the time constraints. Of the remaining two, both Netty and Tomcat would have been adequate, but I felt that the Netty framework and it's nonblocking I/O and threading model, along with the ease of deploying servers built on top of it gave it the edge.

4.3 User Interface

Table 4.3 - User Interface Technology Evaluation

Technology	Established/Stable	Free/OSS	Familiarity
HTML, CSS, Javascript	Yes/Yes	Yes	Low
Sproutcore	No/Yes	Yes/Yes	Low
Adobe Flash	Yes/No	Yes/No	None

Although I briefly considered Flash, it was quickly ruled out for several reasons: 1) It suffers from frequent stability and security issues, 2) It is not open source, 3) I have no familiarity with it, and (with an eye towards the future) 4) It is not available on the majority of mobile computing platforms. If this project is extended to work on tablets or phones, flash involvement would require a complete re-write of the UI. Sproutcore is a javascript based client framework that is used to produce rich-client type of applications inside the browser. Accordingly, the learning curve is steep, and the documentation and support is lesser than is available for simple JavaScript and CSS. Finally, using a simple combination of JavaScript, HTML and CSS is the lowest common denominator and can avoid complicated cross-browser issues. Two libraries, JQuery and JSon-2 provide wrappers and simple API's for use on all four of the major browsers, so that cross-browser issues can be minimized. JQuery is used for form and data binding as well as wrapping the AJAX calls, and Json-2 is used to marshall form encoded data into JSON for transport to the server.

4.4 Monitor Application

The monitoring application will be written entirely in Java, and will utilize several open source libraries for logging and scheduling.

5. ARCHITECTURE

5.1 Database Architecture

The database is a simple setup, running a single database server. The Application server expects properties for the root user, password, host and port, and will configure everything else.

5.2 Server

5.2.1 Data Access Layer

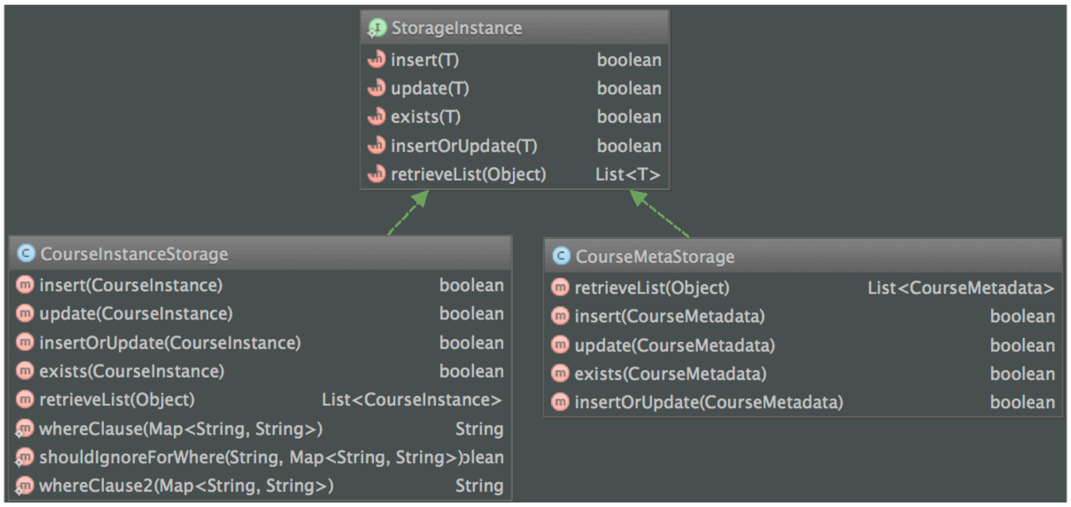
The Application server has an abstracted data API that allows the rest of the application to operate on a set of java objects. The Data Access layer maps these objects to the database tables. The Application server maintains a pool of connections to the database, and executes transactional SQL to insert, delete, fetch and update raw data. This data is then used to create a set of java objects to pass back.

The server defines a simple API for data access for all types of objects, using a generic definition, which each implementation specifies. The Data Storage API returns the implementation for the object type that is requested, and each delegate will perform it’s own transforms and database calls, using predefined SQL strings and JDBC PreparedStatements.

Figure 5.1 - Data Storage Interface Definition

```
public interface StorageInstance<T>{
        boolean insert(T object);
        boolean update(T object);
        boolean exists(T object);
        boolean insertOrUpdate(T object);
        List<T> retrieveList(Object context);
 }
```

Figure 5.2 - UML Class Diagram of Data Access


StorageInstance
insert(T) boolean
update(T) boolean
exists(T) boolean
insertOrUpdate(T) boolean
retrieveList(Object) List<T>
CourseInstanceStorage
insert(CourseInstance) boolean
update(CourseInstance) boolean
insertOrUpdate(CourseInstance) boolean
exists(CourseInstance) boolean
retrieveList(Object) List<CourseInstance>
whereClause(Map<String, String>) String
shouldIgnoreForWhere(String, Map<String, String>)olean
whereClause2(Map<String, String>) String
CourseMetaStorage
retrieveList(Object) List<CourseMetadata>
insert(CourseMetadata) boolean
update(CourseMetadata) boolean
exists(CourseMetadata) boolean
insertOrUpdate(CourseMetadata) boolean

5.2.2 RESTful Application API

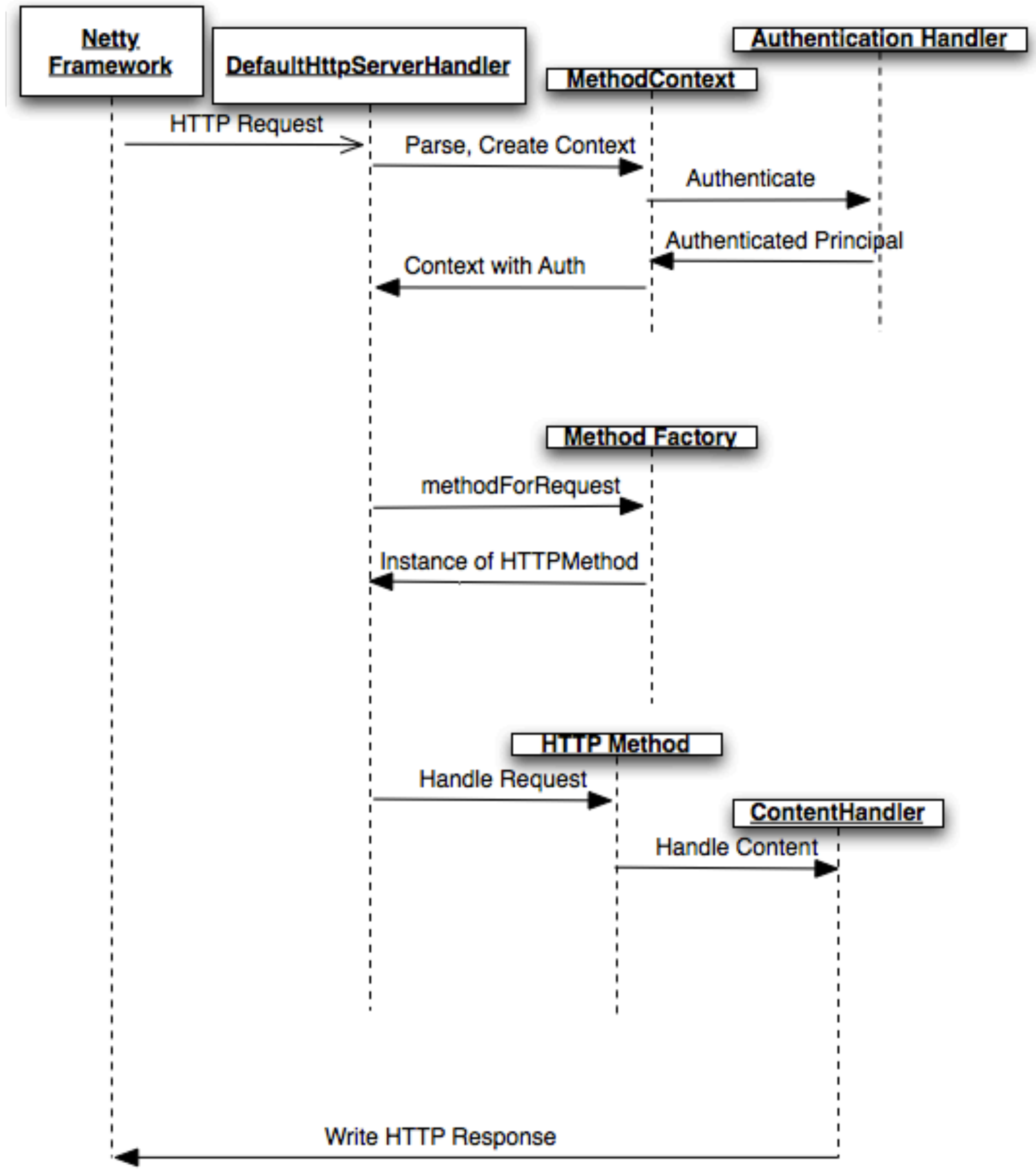
The Application server presents a RESTful API for retrieving and manipulating our data objects. The URL of the request contains a noun and verb combination that is used to indicate the action that is being performed, as well as the type of object that is being operated on. These URL's take the form: http://servername/OBJECT/ACTION - for example: http://servername/classlist/search. A request to this URL would indicate to the server that the client is trying to search for a list of classes, and that the body of the request should contain details about that search (such as search parameters.) A list of objects and the actions, that can be performed, along with example requests is available in Appendix A.

5.2.3 HTTP Handling Architecture

The HTTP part of the NTPAppServer will be handled at a protocol level by the Netty framework, and once the HTTP messsage has been parsed into java objects, then will be handled in a three step process.

1. The incoming HTTP message will be interrogated for pertinent information - Authorization information, target URI and HTTP Method Type.
2. A Factory will create an appropriate HTTP Method Handler, and the HTTP message will be dispatched to that handler.
3. Each handler will, as necessary - determine authorization, parse the content of the request, and either respond, or create a special handler based on the content type.

Figure 5.3 - Sequence Diagram of HTTP POST Method



5.3 User Interface Architecture

The user interface will be composed of 3 screens. Each screen will use HTML for content and form layout, CSS for styling, and JavaScript (both custom and JQuery/JSON libraries) to perform asynchronous(AJAX) requests to the server to post and receive data.

5.3.1 Login Page

The login page will have a HTTP form, with fields for username and password. On submit, a javascript method will send an AJAX request to the server, expecting a response with the landing page URL.

5.3.2 Search Page

The search page will have a HTTP form where the user can specify the requirements for the search: A substring search for class, the term, one course requirement (such as Math or Humanities), and some refinement options. On submit, a javascript method will send this data as JSON to the app server, expecting back fully formatted HTML. When the response comes back, the returned HTML will be inserted into a predefined <div> element at the bottom of the page.

5.3.3 Administration Page

The admin page will be the most complicated page. Initially, the page will only have a single search field in an HTTP form. The user will type a substring to match a set of courses they want to edit (for example, if they wanted to edit all the math classes, they could type MAT and hit enter. It's possible that multiple subjects could match - a search for EN could match English and Environmental Science.) On submit, a javascript method will send an AJAX request with the form's content in JSON format to the server, and will expect a specifically formatted HTML Form to be returned. This form will be inserted into a predefined <div> element in the page. The server will retrieve the appropriate course metadata objects from the database, and create a dynamic form. The returned form will be pre-populated with data parsed from the database and will allow the admin to specify the course requirements fulfilled by each course that is returned in the search. Once complete, the user submits the new form, and it is sent to the server and the Course Metadata table is updated.

5.4 Build and Deployment

The server and monitor processes are built via an Apache Ant script. This script (build.xml) lives at the root of the source tree, and when executed, does the following:

1. Cleans the target directories, and removes old directories
2. Creates directories for compiling and distribution
3. Checks deployment properties for a target platform (Windows/Linux), and generates and increment a build version
4. Compiles both the NTP Application and Monitor Application
5. Creates a deployment directory, and copies all the appropriate code, libraries, static content, properties files and execution scripts, depending on the target platform desired
6. Generates a new manifest file for the Monitor's update job to consume, and inserts the new build number into the properties

The script can perform any of these tasks independently, or for each project, as desired.

6. DATABASE DEFINITIONS AND DATA STRUCTURES

6.1 Setup

The Applications server provides an entry point for creating all the necessary database objects. The user only has to provide a properties file that specifies any overrides to the defaults (Specified in Appendix C)

6.2 Course Metadata

The Course Metadata table is the contextual data that is unique to this application. The NTP admin user is required to update this table and ensure that it has the relevant and correct data. For each course, (for example: English 403 - Children's Literature) there is exactly one entry in this table. There is a unique key that is defined as the tuple between the "subject" (Generally a unique three or four letter combination, such as ENGL or BUSI) and the "course number" (such as 397 or 107L). This table has definitions for which requirements are fulfilled by all instances of this course.

The SQL for this table is available in Appendix E, Figure E.1 - Course Metadata DDL.

6.3 Course Instances

The Course Instance table contains the local representation of all the data that is imported from the WVUP systems. This table represents separate offerings of each class, and all the data about each instance, including time, term, location, start and end dates, instructor and course title.

7. APPLICATION SERVER FUNCTIONS

The application server is the core of the system. It does all the heavy lifting and application functions.

7.1 HTTP/Web Server

The server must perform a substantial subset of the HTTP protocol in order to have proper interaction with the web browsers. In addition to the application functions detailed below, this includes:

- Handling HTTP GET requests for static content. The browser requests varying types of content (HTML, CSS, Javascript, static images) and these must be sent in proper HTTP spec-compliant fashion, as specified in RFC 2616[1]. This requires proper setting of HTTP headers and content.
- Identifying and parsing header values for Cookies and HTTP Authentication.
- Identifying and reading incoming content from HTTP POST requests, as well as parsing it from known forms
- Issuing HTTP Redirects to and from the login forms
- Error handling for internal exceptions, and mapping those to useful HTTP responses

7.2 Authentication & Authorization

The server is primarily set up to handle cookie based authentication. A user logs in from the login page, and their credentials are passed to the authentication hooks.

[1] http://www.ietf.org/rfc/rfc2616.txt

Currently these hooks only look to see if the user is requesting “guest” (student/read only) or “admin” (NTP Staff/read-write) access. Once access is granted, the server creates a session and cookie, and maps the cookie to a local Principal which represents the access granted. It then stores this cookie in a local cache. The cookie is attached to the response, and will be sent on all future requests from the browser until it expires. If a request has a cookie on it, the cookie is fetched from the cache and validated, and the associated Principal is used for the execution of the HTTP Method.

Additionally, code exists (although currently disabled) to allow the use of HTTP Basic Authentication. This type of authentication is useful for mobile devices and other applications that are not browser based.

Inside the application server, each type of action can check the Principal that is associated with the running method. If the action will result in a database write, for example, then the Principal must be a READ/WRITE type, otherwise the server will reject the request with an appropriate error, and redirect the user to the login page.

7.3 Database Access

The Application server maintains a pool of database connections and maps the application requests through to the data access layer. Each request filters through the Database API which uses pre-set SQL statements to create JDBC PreparedStatements and execute them transactionally (with auto-commit). This includes updating course metadata, and retrieving the course information as requested by the student page.

7.3.1 Database Setup

The Application server provides a main method entry point by which all the database configuration and setup of tables and users can be done. Once the database has been installed, this functionality will create all the database schema required for the Application Server to run.

7.4 Information

The application server provides a mechanism to request a base set of information, including uptime, currently running version and the application name via a JSON object.

7.5 Scheduling and Executing Background Jobs

The server implements a scheduling library and schedules a recurring job to query the WVUP servers for any newly published data. If this data is found, a data import of this information is triggered.

7.6 Data Import from WVUP Systems

Course data and information is published by the canonical WVUP systems on a regular basis. This data is accessible via a set of HTML pages that are updated every 5 minutes. These pages are published to the WVUP web site, and are broken down by term and physical location (one page is published for the main campus, and one for the Jackson County campus for each of the three terms, leading to 6 pages in all.) A triggered job retrieves the HTML text and attempts to scrape that data for import into the database. In the event that the data is malformed or unavailable, the job logs an error, and will retry again after a predefined interval. Once the data has been parsed into useful form, it is either inserted or updated in the database (depending on whether the data already exists).

Scraping HTML for this data is potentially very error prone, and can break easily. Since HTML is primarily intended for being viewed, there are many view-specific elements. I have proposed an alternative to IS&T - that they additionally publish this data in a consistent and verifiable format that is intended for machine consumption, preferably JSON or XML.

8. MONITOR APPLICATION FUNCTIONS

8.1 Monitor NTPAppServer Process

Due to the server running on an end-user Windows machine, some of the deployment tools that would normally be available for monitoring and reporting on processes are unavailable. In order to ensure uptime, a scheduled job runs at a regular interval (currently every 3 minutes, but this variable is configurable). This job queries the running host for a list of processes, and then ensures that the NTP app server process is running. If it is not found, then the monitor will attempt to create a new instance of the NTP server, via the command line tools that are provided with the NTPAppServer distribution.

8.2 Update NTPAppServer

The Monitor app performs a critical function: keeping the NTPAppServer updated. The Monitor app is provided a URL to query. At that URL should be a manifest file that contains information about the most recent version of the NTPAppServer that is available, as well as the files that are needed to run that version. The currently running NTPAppServer is then queried via an HTTP call for its currently running version. If the currently running NTP server is out of date, then an update process takes place. This process follows these steps:

1. A temporary file location is created, and the manifest written there.

2. The manifest contains a list of files to download, their locations, and an md5 of each file. The process downloads each file to the temporary location, and checks the md5 against the manifest.
3. All running NTPAppServer processes (usually only 1) are stopped.
4. New NTPAppServer files are moved from the downloaded directory to the NTP Server directory.
5. The new NTPAppServer is started
6. The downloaded files are removed.

8.3 Generate Manifest

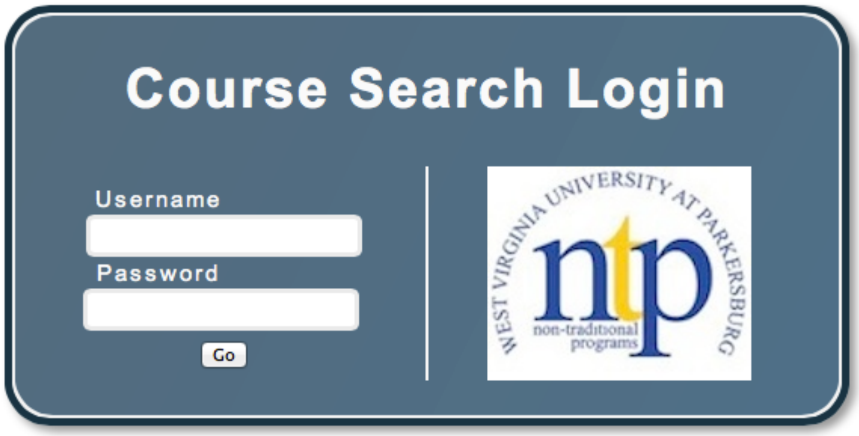
The Monitor app can also be used to create a manifest file for an arbitrary deployment. Given arguments for a directory, version number and a URL, it will generate the manifest for consumption over the web, as required by the update process in section 8.2. An example of a generated manifest is available in Appendix B.

9. USER INTERFACE OVERVIEW

9.1 LOGIN PAGE

The login page has two fields, Username and Password, and a single button to submit the form to the server.

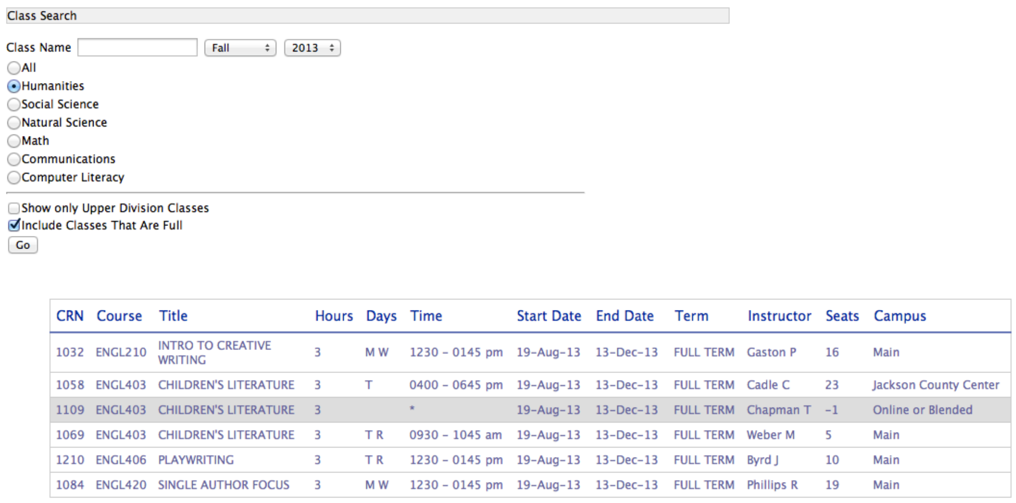
Illustration 9.1 - Login Page UI



9.2 STUDENT CLASS SEARCH PAGE

The class search page is comprised of a form that allows the user to choose the class name (optional), Term (required), a requirement (Optional) and to refine the results.

Figure 9.2 - Class Search Page



Class Search

Class Name [ ] Fall 2013

- ○ All
- ● Humanities
- ○ Social Science
- ○ Natural Science
- ○ Math
- ○ Communications
- ○ Computer Literacy

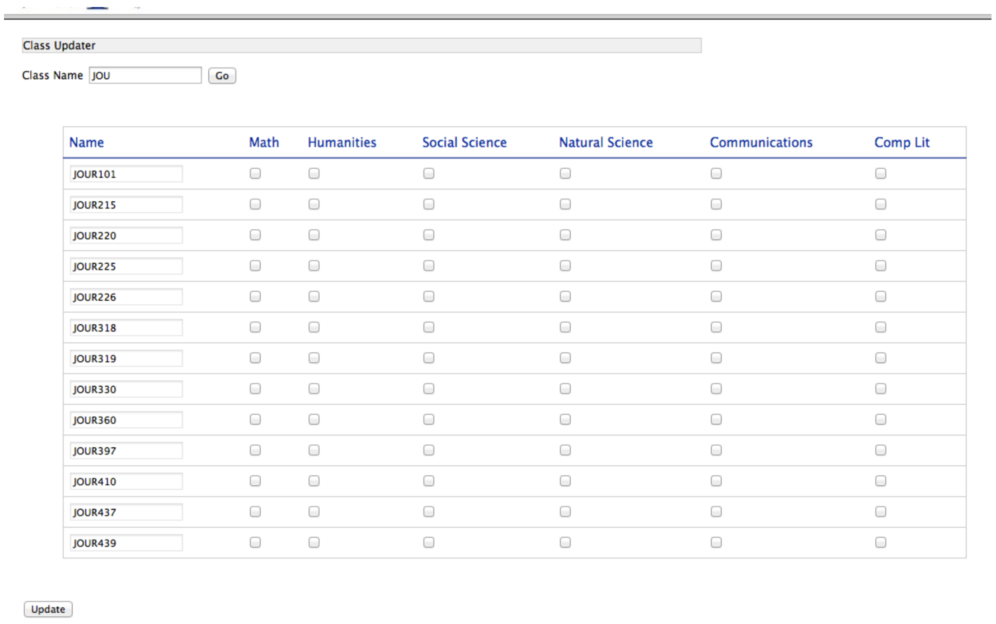
- ☐ Show only Upper Division Classes
- ✓ Include Classes That Are Full

CRN	Course	Title	Hours	Days	Time	Start Date	End Date	Term	Instructor	Seats	Campus
1032	ENGL210	INTRO TO CREATIVE WRITING	3	M W	1230 - 0145 pm	19-Aug-13	13-Dec-13	FULL TERM	Gaston P	16	Main
1058	ENGL403	CHILDREN'S LITERATURE	3	T	0400 - 0645 pm	19-Aug-13	13-Dec-13	FULL TERM	Cadle C	23	Jackson County Center
1109	ENGL403	CHILDREN'S LITERATURE	3		*	19-Aug-13	13-Dec-13	FULL TERM	Chapman T	-1	Online or Blended
1069	ENGL403	CHILDREN'S LITERATURE	3	T R	0930 - 1045 am	19-Aug-13	13-Dec-13	FULL TERM	Weber M	5	Main
1210	ENGL406	PLAYWRITING	3	T R	1230 - 0145 pm	19-Aug-13	13-Dec-13	FULL TERM	Byrd J	10	Main
1084	ENGL420	SINGLE AUTHOR FOCUS	3	M W	1230 - 0145 pm	19-Aug-13	13-Dec-13	FULL TERM	Phillips R	19	Main

9.3 ADMINISTRATOR UPDATE PAGE

The administrator’s update page allows the admin to search for a set of courses, and mass update them with all the requirements that each class fulfills.

Figure 9.3 - Administrator Page



Class Updater

Class Name JOU Go

Name	Math	Humanities	Social Science	Natural Science	Communications	Comp Lit
JOUR101	☐	☐	☐	☐	☐	☐
JOUR215	☐	☐	☐	☐	☐	☐
JOUR220	☐	☐	☐	☐	☐	☐
JOUR225	☐	☐	☐	☐	☐	☐
JOUR226	☐	☐	☐	☐	☐	☐
JOUR318	☐	☐	☐	☐	☐	☐
JOUR319	☐	☐	☐	☐	☐	☐
JOUR330	☐	☐	☐	☐	☐	☐
JOUR360	☐	☐	☐	☐	☐	☐
JOUR397	☐	☐	☐	☐	☐	☐
JOUR410	☐	☐	☐	☐	☐	☐
JOUR437	☐	☐	☐	☐	☐	☐
JOUR439	☐	☐	☐	☐	☐	☐

10. SUMMARY AND RESULTS

10.1 NTP Office

The NTP Office staff has been using the server since the last week of March, and the feedback has been extremely good. They are recommending it to other student advisors on the basis of its ease of use and the amount of time that it is already saving.

10.2 The Monitoring Application

The Monitoring Application has been more complex to write and debug than originally anticipated. Although performing adequately in testing, it is not a substitute for proper process and server monitoring and reporting tools that are available. Although it will suffice for the foreseeable future, if/when the NTP Scheduling Assistant is adopted by a larger audience, proper tools will need to be used to monitor uptime and perform reporting and deployments.

Appendix A - REST Interface Objects & Actions

Table A-1 - Object and Action list.

Object	Available Actions	Purpose
classlist	search, list	Searches for or lists class instances
classmeta	search, update, list	Searches for, lists or updates class metadata
authentication	validate, invalidate	used to validate or invalidate credentials or cookies.
info	list	Returns simple information about the server

A.1 Example Requests

A.1.1 Info Request

```
> GET /info/list HTTP/1.1

{"application":"NTPAppServer","uptime":
3310,"version":"1"}
```

A.1.2 Authentication Requests

A.1.2.1 - Validation Request/Response

Note the username/password in the body, and the relative URL and Set-Cookie header in the response.

```
Request:
POST /authentication/validate HTTP/1.1
Host: localhost:8080
Content-Length: 65
Content-Type: application/json; charset=UTF-8

[{"name":"user","value":"admin"},
{"name":"pass","value":"admin"}]


Response:
HTTP/1.1 200 OK
Content-Type: text/plain; charset=UTF-8
Connection: keep-alive
Set-Cookie: NTP=04c444cb-b071-47a4-9307-12ed84889358;
Expires=Fri, 26-Apr-2013 14:34:58 GMT; Path=/
Content-Encoding: gzip
Content-Length: 43

/index-admin.html
```

A.1.2.2 - Invalidation Request/Response

A logout that re-directs to the login page, and tells the browser the cookie is expired.

```
Request:
GET /authentication/invalidate HTTP/1.1

Response
HTTP/1.1 301 Moved Permanently
Content-Length: 11
Location: /unauthorized/login.html
Set-Cookie: NTP=expired; Expires=Tue, 23-Apr-2013
22:52:48 GMT; Path=/

File Moved.
```

A.1.3 Sample Search

A sample search for Upper Division Humanities classes in Fall of 2013.

```
Request:
POST /classlist/search HTTP/1.1
Host: localhost:8080
Content-Length: 169
Content-Type: application/json; charset=UTF-8
Cookie: NTP=04c444cb-b071-47a4-9307-12ed84889358

[{"name":"classname","value":""},
{"name":"term","value":"Fall"},
{"name":"year","value":"2013"},
{"name":"searchtype","value":"humanities"},
{"name":"upperdiv","value":""}]
```

And the response, which contains dynamic HTML in a table.

```
Response:
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Set-Cookie: NTP=04c444cb-b071-47a4-9307-12ed84889358
Content-Encoding: gzip
Content-Length: 467

<table id="gen-table">
      <thead>
            <th scope="col">CRN</th>
            <th scope="col">Course</th>
            <th scope="col" width="150px">Title</th>
            <th scope="col" width="40px">Hours</th>
            <th scope="col">Days</th>
            <th scope="col">Time</th>
            <th scope="col">Start Date</th>
            <th scope="col">End Date</th>
            <th scope="col">Term</th>
            <th scope="col">Instructor</th>
            <th scope="col">Seats</th>
            <th scope="col">Campus</th>
      </thead>
      <tbody>
            <tr BGCOLOR="#FFFFFF">
                  <td>1069</td>
                  <td>ENGL403</td>
                  <td>CHILDREN'S LITERATURE</td>
                  <td>3</td>
                  <td>T R</td>
                  <td>0930 - 1045 am</td>
                  <td>19-Aug-13</td>
                  <td>13-Dec-13</td>
                  <td>FULL TERM </td>
```

```
                <td>Weber M</td>
                <td>3</td>
                <td>Main</td>
            </tr>
        </tbody>
    </table>
```

Appendix B - Sample Manifest for Web Update.

```
{
   "entries": [
      {
         "file": "/dbmigrate.sh",
         "hash": "1a427457d34769877cc7373124e6f5e",
         "size": 372,
         "time": 1366910201000,
         "type": "NEW"
      },
      {
         "file": "/jars",
         "hash": "",
         "size": 0,
         "time": 1366910201000,
         "type": "DIRECTORY"
      },
      {
         "file": "/jars/commons-codec-1.7.jar",
         "hash": "e47ef8e1a0c11ee7e41704816cda890",
         "size": 259600,
         "time": 1366910201000,
         "type": "NEW"
      },
      {
         "file": "/start-server.bat",
         "hash": "956145c527c7d986adea92aea082e5",
         "size": 362,
         "time": 1366910201000,
         "type": "NEW"
      }
   ],
   "guid": "4be78880-f68c-4981-b488-b98afbd08b58",
   "time": 1366910202258,
   "urlRoot": "http://localhost:8000/",
   "version": 1098
}
```

Appendix C - List of External Libraries Used

Table C.1 - External Library List

Library	Source	Use
Netty 3.6.2	http://netty.io	IO/Web Framework, Server base.
Apache Commons	http://commons.apache.org	Various utilities.
SLF4J	http://www.slf4j.org	Unified Logging Facade
Quartz 2.1.6	http://www.quartz-scheduler.org	Task Scheduling framework
Logback 1.0.9	http://www.quartz-scheduler.org	Logging Implementation
Json-Smart 1.1	http://code.google.com/p/json-smart/	JSON Creation & Parsing
postgreSQL JDBC	http://www.postgresql.org	Bridge for JDBC Database Access

Appendix D - Database Properties

These are properties that can be specified for the NTPAppServer to use in configuring it's database:

Table D.1 - Database Properties

Property	Default Value
db.host	localhost
db.root.username	postgres
db.root.password	postgres
db.name	course_schedules
db.readwrite.username	cs460
db.readwrite.password	cs460
db.readonly.username	cs460
db.readonly.password	cs460
db.connectionpool.max.size	10

APPENDIX E - DATABASE TABLES

Figure E.1 Course Metadata DDL

```
CREATE TABLE course_meta
(
  subject character varying(8) NOT NULL,
  course_number character varying(8) NOT NULL,
  humanities boolean NOT NULL DEFAULT false,
  natsci boolean NOT NULL DEFAULT false,
  socsci boolean NOT NULL DEFAULT false,
  math boolean NOT NULL DEFAULT false,
  communications boolean NOT NULL DEFAULT false,
  complit boolean NOT NULL DEFAULT false,
  upperdiv boolean NOT NULL DEFAULT false,
  CONSTRAINT course_meta_pkey PRIMARY KEY (subject,
course_number)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE course_meta
  OWNER TO cs460;
```

Figure E.2 Course Instance DDL

```
CREATE TABLE course_instance
(
  crn character varying(12) NOT NULL,
  type character varying(8) NOT NULL,
  crosslisted boolean NOT NULL DEFAULT false,
  subject character varying(8) NOT NULL,
  course_number character varying(8) NOT NULL,
  course_title character varying(255) NOT NULL,
  credits smallint NOT NULL,
  days character varying(16) NOT NULL,
  "time" character varying(32) NOT NULL,
  instructor character varying(64) NOT NULL,
  room character varying(32) NOT NULL,
  start_date timestamp without time zone NOT NULL,
  end_date timestamp without time zone NOT NULL,
  seats_available smallint NOT NULL,
  term_length character varying(64) NOT NULL,
  campus character varying(32) NOT NULL,
  term character varying(8) NOT NULL,
  year character varying(4) NOT NULL,
  CONSTRAINT course_instance_pkey PRIMARY KEY (crn)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE course_instance
  OWNER TO cs460;
```