|  |
| --- |
| University of MISSOURI-kansas city |
| SternerLearn |
| Increment 1 Report |
|  |
| **Connor Ledgerwood and Devin Turner** |
| **3/12/2013** |

|  |
| --- |
|  |

# Design

TBD

# Implementation

For the first increment, we implemented the bulk of the first two web services, along with the underlying database structure. We purposefully created this backend first, so that we could begin work on the UI without interruption, and to allow more thorough testing of the individual services and UI functionality.

## Database

The below describes the implementation of the database tables and columns. We designed the database to allow simple links between tables using unique identification numbers for different tables, such as account IDs and assignment IDs.

### Accounts

**id(int)**

Identification number of this account. Unique for all accounts.

**accountType(int)**

Type of the account. Either parent, student, or teacher.

**linkedID(int)**

The linked ID. For a parent, this must be a student ID, and vice versa. Teachers do not have linked IDs.

**emailAddress(string)**

The email address associated with the account.

**userName(string)**

The user name associated with the account.

**password(string)**

The password associated with the account.

**realName(string)**

The user’s real name.

### Assignments

**assignmentID(int)**

Identification number for this assignment. Unique for all assignments.

**courseID(int)**

The identification number of the course this assignment is for.

**name(string)**

Name of the class.

**description(string)**

More description of the class, including prerequisites, etc.

**totalPoints(float)**

Total points this assignment is worth.

**dueDate(datetime)**

Date and time the assignment is due.

**isTest(bit)**

Boolean value which indicates whether this is a test type.

### Courses

**courseID(int)**

Identification number for this course. Unique for all course.

**staffID(int)**

ID number of the teacher for this course.

**className(string)**

The name of the course.

### Grades

**gradeID(int)**

Identification number for this grade. Unique for all grade.

**assignmentID(int)**

ID number of the assignment this grade is for.

**studentID(int)**

ID number of the student this grade is for.

**pointsEarned(float)**

Total points the student received for this assignment.

**dateSubmitted(datetime)**

Date the student submitted the assignment. Can be used to determine lateness of the submission.

### Infractions

**infractionID(int)**

Identification number for this infraction. Unique for all infraction.

**studentID(int)**

ID number of the student this infraction is for.

**infractionType(int)**

Type of infraction, such as tardiness, absence, suspension, etc.

**description(string)**

More details the teacher can add to describe the infraction.

### Locations

**latitude(int)**

Latitude of the location.

**longitude(int)**

Longitude of the location.

**studentID(int)**

Student ID of the student this location was recorded for.

**time(datetime)**

Time the location was recorded.

### Meta

This table contains all of the data other tables may need when creating new rows, in addition to the user input. For now this only contains the “last ID” of the different types of tables. This is used to make the creation of new IDs much more efficient.

**lastAccountID(int)**

The last account ID number that we generated.

**lastCourseID(int)**

The last course ID number that we generated.

**lastAssignmentID(int)**

The last assignment ID number that we generated.

**lastGradeID(int)**

The last grade ID number that we generated.

**lastInfractionID(int)**

The last infraction ID number that we generated.

**lastLocationID(int)**

The last location ID number that we generated.

**lastNotificationID(int)**

The last notification ID number that we generated.

### Notifications

**notificationID(int)**

Identification number for this notification. Unique for all notification.

**description(string)**

Text that was contained in the notification.

**guardianID(int)**

Guardian that received the notification.

**teacherID(int)**

Teacher that sent the notification.

### Students

For now, we have hardcoded the number of classes to be 3. However, as a low priority task we have noted the potential for modifying the number of classes. This would probably be done by setting an entry in the Meta table to “numClasses” and then dynamically creating new columns named {class1..class(numClasses)}.

**id(int)**

Account ID for this student.

**class1(int)**

Class ID for their first class.

**class2(int)**

Class ID for their second class.

**class3(int)**

Class ID for their third class.

## AccountService

Most of these functions should be relatively self explanatory, but notes are below functions which were deemed necessary.

public string Register

(

string name, // Username of the account holder

string email, // Email address

string password, // Password

string passwordRepeat, // Password confirmation

int type // Type of account to create

)

public string Login

(

string email, // Email address for the account

string password // Password

)

public string LinkAccounts

(

string parentEmail, // Email address for the parent's account to link

string parentPassword, // Password for the parent

string studentEmail, // Email address for the student's account to link

string studentPassword // Password for the student

)

LinkAccounts unsurprisingly links two accounts, if they are a parent and teacher account. This link is used to determine which students should be associated with which guardians.

public int generateNewID

(

ID\_TYPE type // Type of table (corresponding with an enum value) to generate an ID for

)

generateNewID is not a web service, but it is widely used by other web services to generate new IDs.

## StudentDataService

This is the largest web service planned for our application. This is where all of the school data is retrieved and added: classes, assignments, grades, etc. To allow easier manipulation, several serializable classes were created, which effectively correspond to their SQL tables.

### Assignment

// Correspond to C# equivalents of the Assignments columns

public int mAssignment;

public int mCourseID;

public string mName;

public string mDescription;

public float mPoints;

public DateTime mDueDate;

public bool mIsTest;

### Infraction

// Correspond to C# equivalents of the Infractions columns

public int mInfractionID;

public int mStudentID;

public StudentDataService.INFRACTION\_TYPE mInfractionType;

public string mDescription;

public DateTime mDate;

### Grade

// Correspond to C# equivalents of the Grades columns

public int mGradeID;

public int mAssignmentID;

public int mStudentID;

public double mPointsEarned;

public DateTime mDateSubmitted;

### Functions

public bool addClass

(

int aTeacherID, // Teacher ID to verify and to have teaching the class

string aPassword, // Teacher's password

string aClassName // Name of the class to add

)

public bool addStudentToClass

(

int aTeacherID, // Teacher ID used to verify

string aPassword, // Teacher's password

int aClassID, // Class ID to add student to

int aStudentID // Student to add

)

public List<int> getClasses

(

int aStudentID // Student ID to get classes for

)

This retrieves the classes for a student. Currently it simply returns the ID numbers of the classes, but we plan to modify this to return Class objects, like getAssignments below, to make it more robust.

public List<Assignment> getAssignments

(

int aClassID // Class ID to get assignments for

)

This returns a list of Assignment objects. Because Assignments are serializable and we have the GenerateScriptType on our WebService, these should be readable from other languages’ web service calls.

public bool addGrade

(

int aTeacherID, // Teacher ID to verify

string aPassword, // Password for teacher to verify

int aStudentID, // Student ID to add grade for

int aAssignmentID, // Assignment ID to add grade to

double aPointsReceived, // Points the student received

DateTime aDateSubmitted // Date the student submitted it

)

public List<Grade> getGrades

(

int aStudentID, // Student ID to get grades for

int aClassID // class ID to get grades for

)

public bool addInfraction

(

int aTeacherID, // Teacher ID to verify

string aPassword, // Password to verify

int aStudentID, // Student ID to add infraction to

INFRACTION\_TYPE aInfractionType, // Type of infraction

string aDescription, // Description of infraction

DateTime aDate // Date the infraction occurred

)

public List<Infraction> getInfractions

(

int aStudentID // Student ID to get infractions for

)

public int getParentID

(

int aStudentID // Student ID to get the parent for

)

public int getStudentID

(

int aParentID // Parent ID to get student for

)

protected bool verifyTeacher

(

int aTeacherID, // Teacher ID to verify

string aPassword, // Teacher's password

SqlConnection aConnection // Connection to use. Caller must open and close it

)

This is not a web service, but it is widely used. For all of the “addX” services, we call this to verify that the teacher’s identification is valid. This is used to prevent other users from accessing functionality which should only be available for teachers.

protected bool generateGenericInsertCommand

(

SqlConnection aConnection, // Connection to execute command with

params string[] aList // Variable length list of parameters

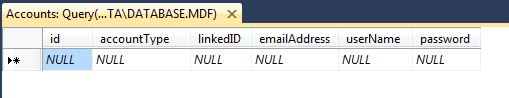
)

This is not a web service, but it is used for most of the “addX” services. It was used to make generating the command string in a simpler fashion.

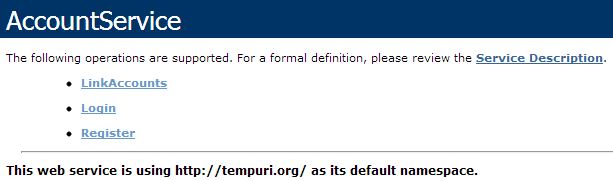
# Testing

Because all of our services for this iteration were web based services, we did not use NUnit for our testing. Instead, we tested our services manually using the web-based interaction and examining the database contents.

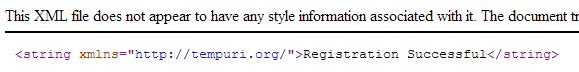
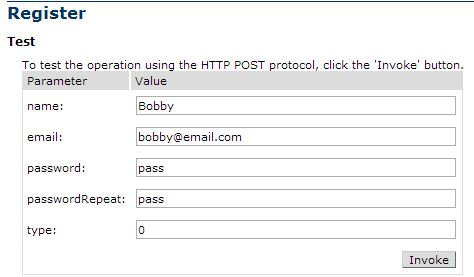
Account database is initially empty.



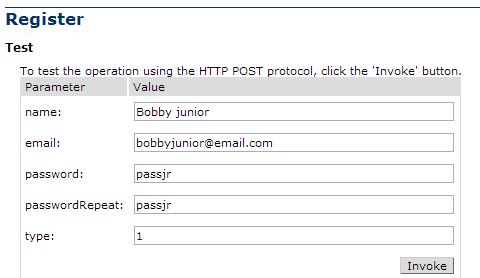
Here is the list of services available on the AccountService class. This file handles account creation and logging in.



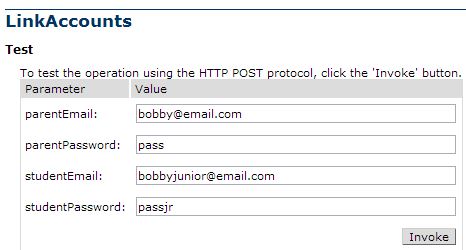
A parent account 'Bobby' is created.

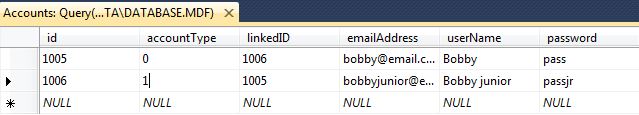


A student account 'Bobby junior' is also created. This will also add him to the Student database.

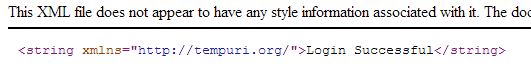


Now the two accounts are linked.

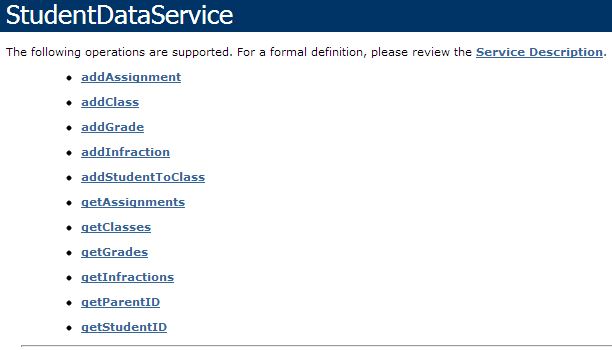


The results are shown in the Accounts database.

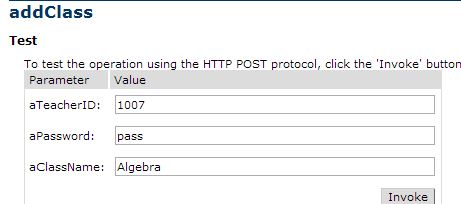
Now either account can log in. Note that all methods will return a non-string acknowledgment after a browser interface has been implemented.



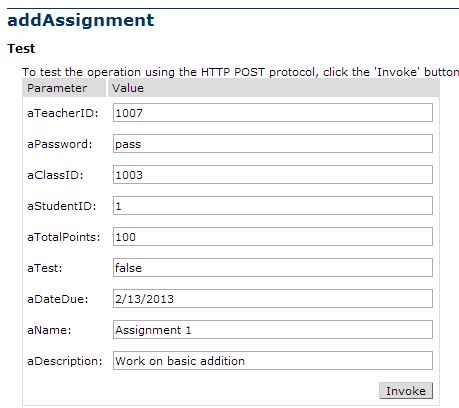
Here is the list of services available on the StudentDataService class. This file is mostly concerned with managing classes and other data relating to a student account.



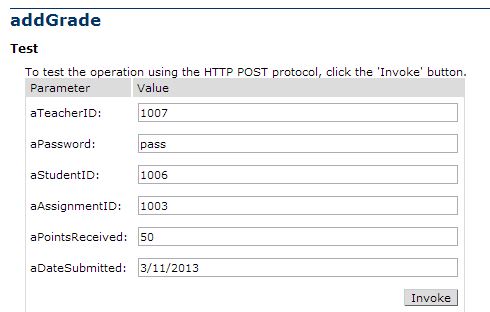
First, a class 'Algebra' is created. The teacher account was already created.

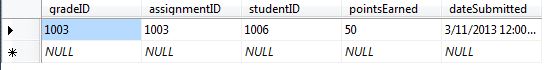
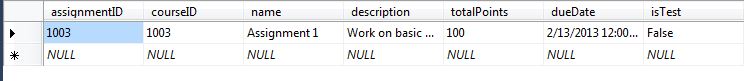
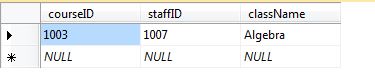


Then an assignment is added to the course.



Next, a grade for Bobby Junior is added to the Grades database.



All of the information has now been added to the appropriate database.

# Project Management

We have no issues or concerns with our project’s progress so far. We were able to complete virtually all of the tasks that were originally planned for Iteration 1. The only tasks we did not complete were either determined unnecessary, or we decided it would be more appropriate to work on at a later date. In addition, we completed several tasks from Iteration 2, which opens up additional time in our schedule to potentially work on some additional features.

## Work Completed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Product | Iteration | | | Task | User | Spent effort (hours) |
| SternerLearn | | | Iteration 1 | Create Accounts Database table | Devin Turner | 0.50 |
| SternerLearn | | Iteration 1 | | Create Assignments Database table | Devin Turner | 0.50 |
| SternerLearn | | Iteration 1 | | Create Classes Database table | Devin Turner | 0.50 |
| SternerLearn | | Iteration 1 | | Create Grades Database table | Devin Turner | 0.50 |
| SternerLearn | | Iteration 1 | | Create Infractions Database table | Devin Turner | 0.50 |
| SternerLearn | | Iteration 1 | | Create Iteration 1 Report | Devin Turner | 4.00 |
| SternerLearn | | Iteration 1 | | Create Meta Database table | Devin Turner | 0.50 |
| SternerLearn | | Iteration 1 | | Create Notifications Database table | Devin Turner | 0.50 |
| SternerLearn | | Iteration 1 | | Create Students Database table | Devin Turner | 0.50 |
| SternerLearn | | Iteration 1 | | Create UML documentation | Connor Ledgerwood | 6.00 |
| SternerLearn | | Iteration 1 | | End-to-end testing | Connor Ledgerwood | 6.00 |
| SternerLearn | | Iteration 1.5 | | Create AccountManagement::Login Web Service | Connor Ledgerwood | 4.00 |
| SternerLearn | | Iteration 1.5 | | Create AccountManagement::Register Web Service | Connor Ledgerwood | 4.00 |
| SternerLearn | | Iteration 1.5 | | Create StudentData::addAssignment Web Service | Devin Turner | 2.00 |
| SternerLearn | | Iteration 1.5 | | Create StudentData::addClass Web Service | Devin Turner | 2.00 |
| SternerLearn | | Iteration 1.5 | | Create StudentData::addStudentToClass Web Service | Devin Turner | 2.00 |
| SternerLearn | | Iteration 1.5 | | Create StudentData::addTeacher Web Service | Devin Turner | 2.00 |
| SternerLearn | | Iteration 1.5 | | Create StudentData::getAssignments Web Service | Devin Turner | 2.00 |
| SternerLearn | | Iteration 1.5 | | Create StudentData::getClasses Web Service | Devin Turner | 2.00 |
| SternerLearn | | Iteration 2 | | Create StudentData::addGrade Web Service | Devin Turner | 2.00 |
| SternerLearn | | Iteration 2 | | Create StudentData::addGuardian Web Service | Connor Ledgerwood | 2.00 |
| SternerLearn | | Iteration 2 | | Create StudentData::addInfraction Web Service | Devin Turner | 1.00 |
| SternerLearn | | Iteration 2 | | Create StudentData::getGrades Web Service | Devin Turner | 2.00 |
| SternerLearn | | Iteration 2 | | Create StudentData::getInfractions Web Service | Devin Turner | 1.00 |
| SternerLearn | | Iteration 2 | | Create StudentData::getParentID Web Service | Devin Turner | 1.00 |
| SternerLearn | | Iteration 2 | | Create StudentData::getStudentID Web Service | Devin Turner | 1.00 |

## Work to be Completed

There are only two tasks in Iteration 1 which were not completed. Both of these tasks are part of the Text Message feature, which will allow guardians to view their student’s text messages. Because we have not begun development of our mobile application for Iteration 1, we decided to wait to implement the Text Message database table and corresponding web services until then. Once we determine exactly what information Android will be able to provide about the text message data, we will be in a better position to create the table and services.

# Deployment

## Project Website

<http://170.224.169.101/lab5/main.html>

## Agilefant

<https://cloud.agilefant.org/evaluator102/login.jsp>

To log in, use the username “professor” and password “password”.

## Web Services

## GitHub

<http://github.com/clkv5/cs551_project/>