Specification: *Daily Information App*

Form and Behavior Specification

Table of Contents

Reviewed?

Revision	Date	Who
	8/1/17	National Instructor Brad
	8/1/17	Head Instructor Tom

Revision History

Revision	Date	What	Who
	8/5/17	Added Table Definitions & API	HI Tom
	9/3/17	Minor updates to Table Definitions - added table names, Minor updates to API - deleted oyd_id for attendance functions, Added Section 5: Phase ideas, added class_group to Informations Table	HI Tom

1. Overview

- *This document describes a browser-based "app" for an instructor to enter in daily metrics to track school activity and monitor school growth. At the end of each day, a designated instructor for each school can enter this daily information to keep track of: attendance, prospective students (information), new students, new courses (e.g. MLT Training such as championships, iron hand, week long seminar, week end seminar, etc.), tests, drops, etc. By entering this data for each school on a daily basis, the instructors within the school can monitor school growth. With each school's information up to date, a Regional instructor can monitor regional growth, and a higher belts can observe general school functions nationally.*

1.1. Terminology

Information - the school term for prospective student. As used herein, it means someone who has reached out to the school to learn about training. It is differentiated from a Lead. Leads are *not tracked in this system, since it will make more sense to keep leads in a mail-list type system such as intercom.io or mailchimp.*

Event - for the purposes of this document, the term event pertains to an activity that occurs within the school that warrants data capture. In phase 1, the events include: information, new student, new course, test, drop

DataStudio - Google's tool for graphing data, and developing custom reports which can be shared to specific users. (https://datastudio.google.com)

1.2. Desired Outcome

By having instructors capture standardized metrics for every school, all stakeholders will be better informed regarding school health and growth. School-level instructors can better monitor student attendance achievement, regional instructors can monitor growth of the region and the schools within it, and look for both positive and negative trends to balance, and Oom Yung Doe can better see how things are progressing Nationally and beyond, over time.

1.3. Supporting Artifacts

The following artifacts can be used as reference to this document.

The following is a video animatic of the general functionality on the data input side.
https://youtu.be/UQoq28DLjg8

More to come with respect to database table structures and data visualization

2. Interaction Detail

This section describes specific interactions.

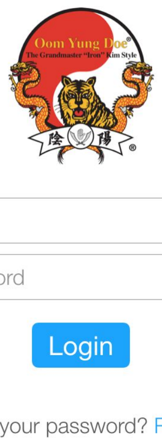
2.1. Instructor Experience

Login

For the phase described within this document, only a designated instructors need access to the web app for data entry. Other instructors could see reports via Google Data Studio directly.

Each school would designate an instructor or team of instructors responsible for making sure that daily information was entered into the system. An instructor on this team would log into the app via a web browser. It would be possible to log in via a phone, but much more convenient to use a device with a keyboard and larger monitor.

The instructor would go to the app URL (TBD) and enter their credentials in the form of an email and a password. We can add an reset password feature, but this may not be necessary in Version 1 - we can manually reset passwords if needed.

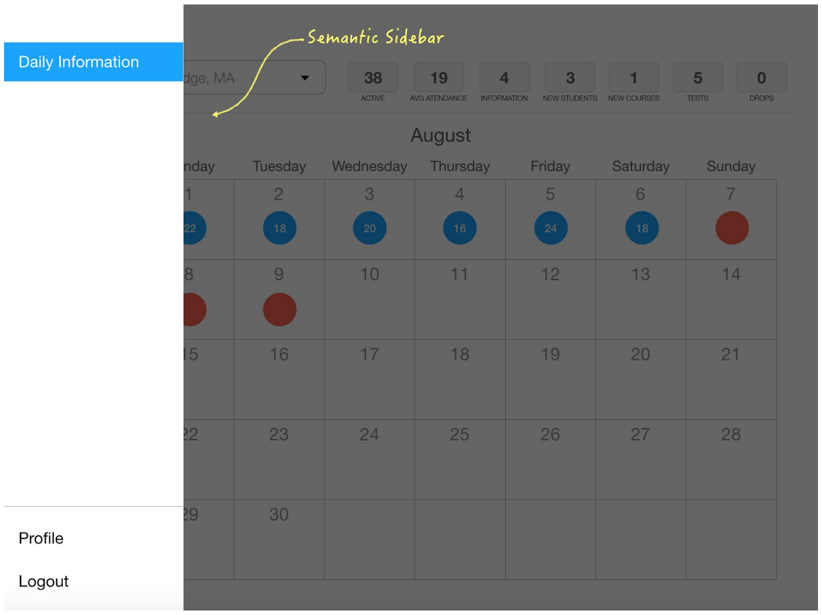


Oom Yung Doe
The Grandmaster "Iron" Kim Style
email
password
Login
Forgot your password? Reset it.

Login Screen. Instructor would enter email address and password. For resetting the password, instructor would enter email address in separate modal. The password would be reset to a temporary password, and emailed, or some other common mechanism for resetting the password.

Navigation Menu

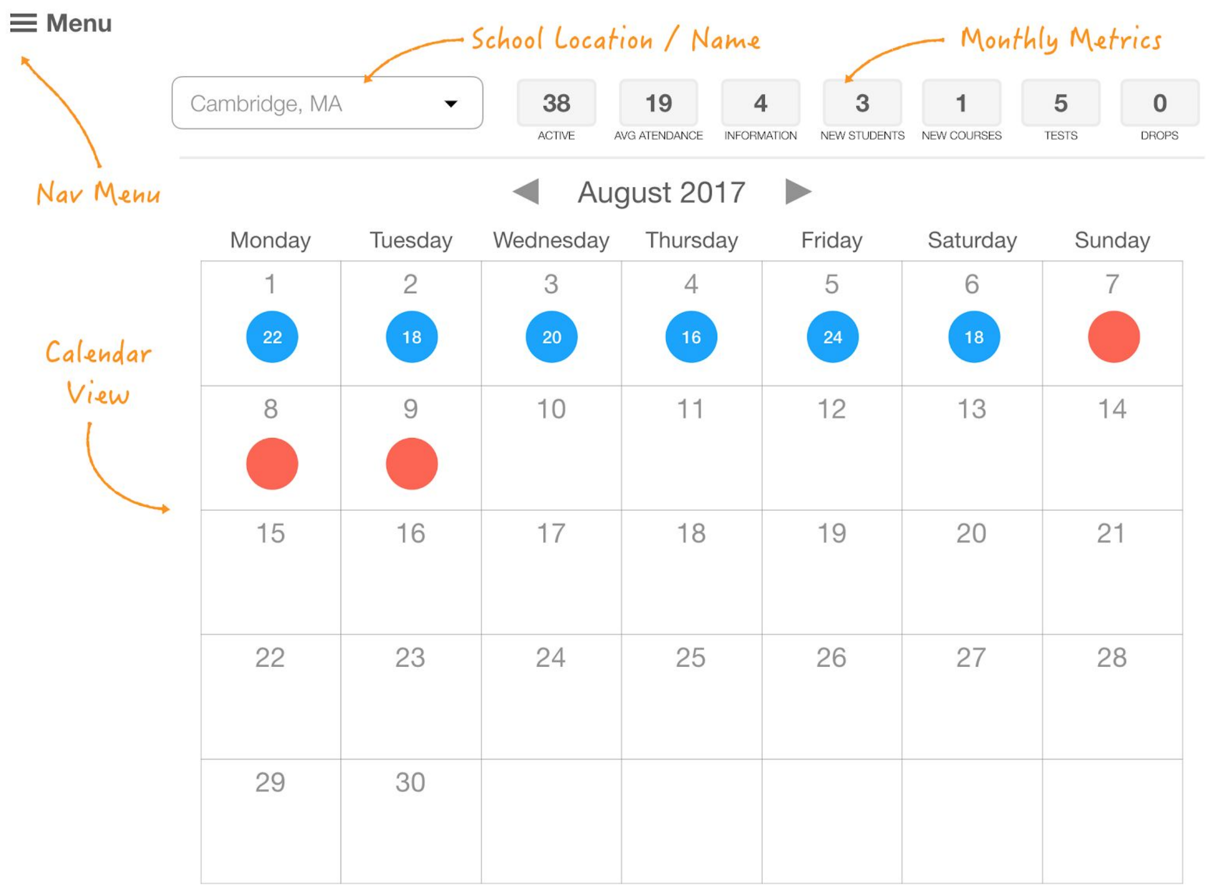
In the first iteration of the app, we may not need a navigation menu of any significance, since we will only initially have one function - the daily information entry. However, in the mock-ups, a nav is provided using a standardized nav slide out.



Nav Menu. Instructor can choose Daily Information, Profile and Logout in initial phase. Other “apps” can be added over time.

Calendar View

The intent of the calendar view is to give a visual to each school’s data tracker (or tracking team) an indication of data being entered, or as important, lack of data being entered. By showing the view as a calendar, the emphasis is on recording the appropriate school data events every day.



Calendar View. Instructor can see days of month, and get indication of which days need data entry.

Key aspects of the Calendar View

1. Navigation to sidebar menu, for other functionality in the future, as well as basics such as profile, log out.
2. School location indication AND / OR navigation. Most users who log in will likely have access to a single school account. But if the user is permissioned for multiple accounts, the list of schools permissed appears in the drop down.
3. Monthly Metrics - as a proxy for a monthly report, metrics are tallied automatically from the daily sheets to provide a real-time status of the month in progress, or prior months for past data. These metrics are straight sums of the daily sheet information, with the exception of two metrics:
    a. Avg. Attendance: which is the sum of the attendance entered for the month divided by the number of days entered.
    b. Active: this is sum of both adults and children from the active student table. (TBD by HI Tom).

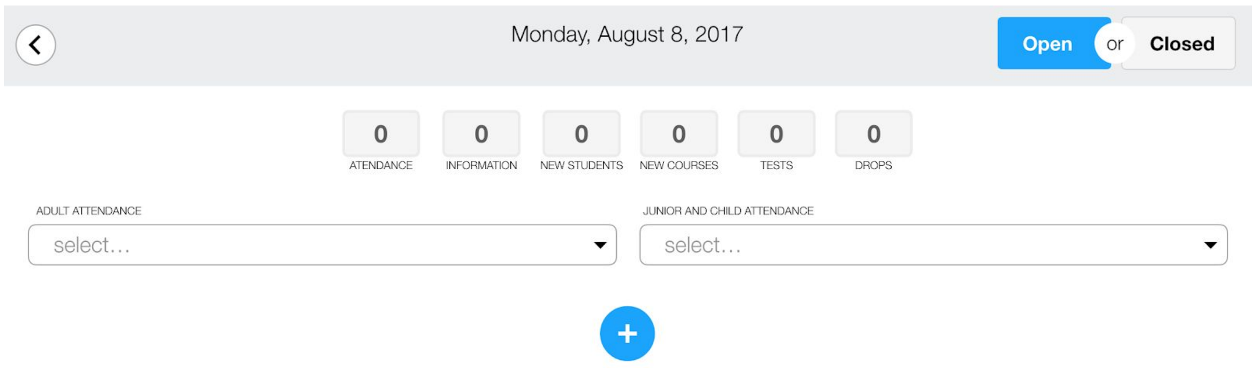
NOTE: The design does not differentiate between adults and children for these metrics. We will show variances in a monthly report via Data Studio that will provide more information with respect to Adults, Juniors and Children.

4. Calendar View.
    a. The month is displayed at the top, with an option to navigate to prior months. There is no need to navigate to the future, although it is represented in the mock-up so that one could see how one could navigate forward in time if one was reviewing historical data.
    b. Blue dot indicates school was open, and number within indicates attendance. We will keep track of instructors within a school, so attendance will always be at least one if the school was open.
    c. Red dot indicates data has not been entered for the day.
    d. Grey dot indicates the school was closed.
    e. Clicking on a day navigates the user to the Daily Information Sheet where attendance and events are added

Daily Sheet

Initial View

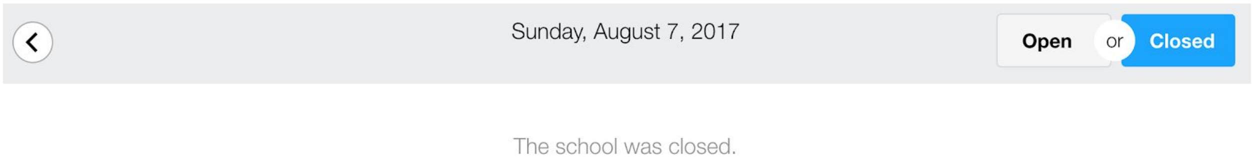
When first entering into a daily sheet for data entry, the view contains no data, as follows:



Empy Daily Sheet. Instructor navigates to particular day to enter data. Left arrow at top navigates back to Calendar.

School Closed

On days when the school is closed, the instructor can toggle the school Open / Closed input. Mock-ups shown use Semantic UI toggle. If toggled to closed, the data entry container changes to indicate that the school was closed, as follows:



School Closed Daily Sheet. Simple view that states the school was closed.

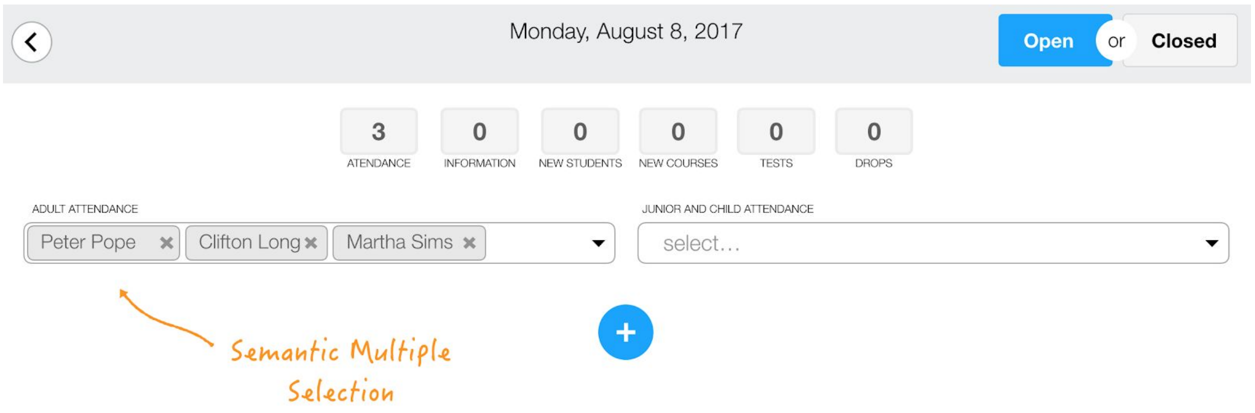
School Closed Event: When an instructor selects the School Closed toggle, we need to write an event into the database that indicates the school was closed. In this way, we can easily track that data was entered, or not entered, for every day for every school.

School Open - Attendance

When the school is open, the instructor tracks attendance, and all relevant daily information.

To track attendance, the instructor clicks on a multiple select drop down, and selects every active member of a school (instructor or student) who attended the school on the given day.

Shown in the mock-up, there are two dropdowns: Adults and Junior / Children. The reason is that seeing a list of both adults and children is more difficult to navigate for an instructor. The proposed mode of providing the active Adults vs Juniors / Children is via a back end table of Active that has a column for student type (Adult, Child, Junior) that is strictly based on age, calculated from a required field of birthday. The reason for this is that the status will change over time, and individual schools may classify certain students one way, and other students another way. We need a national standard for tracking, even if there are minor variations by school. And we need to minimize the amount of upkeep for instructors - thus the calculation of status.



Daily Sheet - Entering Attendance. Instructor uses multi-select from active lists.

School Open - Events

At the core of daily data are the events that we want to capture. All events are on a per-person basis. So for example, if three students test on a given day, this results in three events, one for each student. If two information come in in a given day, it results in two events.

The initial set of events we've discussed are as follows:

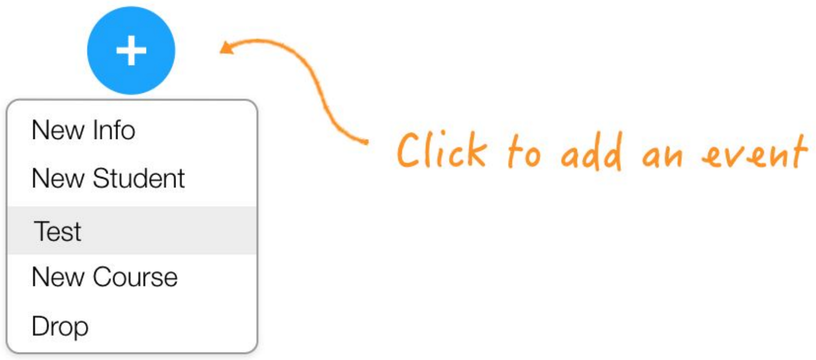
- Information - Whenever a school has a prospective student that has reached out in some manner to learn more about training, we track this individual as an *information*. This is differentiated from a Lead, which is someone who is less likely to train, but that we still want to send messages to, etc. A lead may be someone who signs up to win a free lesson at town event, or someone an instructor bumps into while flyering, whereas an information is someone who took an active effort to reach out the school. We dod not need to capture leads in this app, because most likely the local area will want to enter leads into a correspondence system like intercom.io or mailchimp. At some point in the future, we may wish to add leads and integrate with such systems. Regarding information, we want to collect basic contact information, age, occupation, etc. and we want to a status: walk-in, contacted. We also want to track source, which would be swalked by, social media, website, flyer, event.
    - The fields captured are:
    - Additional fields stored in the db are:
- New Student. This occurs when someone signs up and makes a payment. Someone will have to have been an information to reach this stage. Even someone who comes in off the street and immediately signs up is first designated and information then a new student. In this way, selection of anyone who will be a New Student comes from a drop down of Information and Drops, which cover all the use cases for someone signing up.
    - The fields captured are:
    - Additional fields stored in the db are:
- New Course. This occurs when someone signs up for MLT training. We will designate a list of types, which should include: Chung Dan Hyung Championship, Chung Jun Hyung Championship, Full Body Conditioning, Iron Hand Level 1, Bagwa Level 1, Bagwa Level 2, Week End Seminar (1 Day), Week End Seminar (2 Day), Week Long Seminar, Week Long Seminar Partial. We can also allow for a duration and a description.
- Test. This occurs when someone tests. Name, rank, pass/fail, as shown in the mock-up.
- Drop. This occurs when someone drops.
- New Position. We can considering adding an event for a new position / promotion.

On any given day, any number of events (from 0-n) can be added. Once added and saved, any event can be edited and/or deleted.

We will need to discuss how often the DB is backed up, and where.

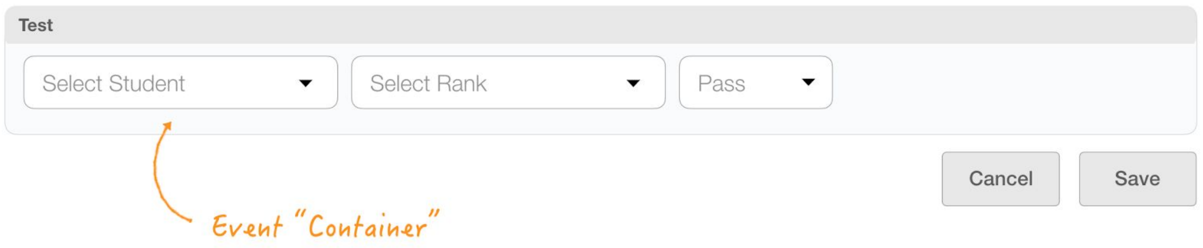
Adding Events - Flow of UI

An instructor will click the blue plus button to add an event.



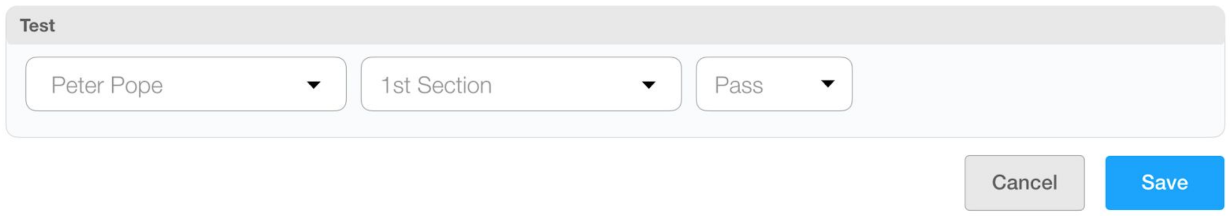
Daily Sheet - Entering Events. Instructor selects an event to add.

The Container for the Event will appear, which will provide an interface for adding the particular event. The Test container is shown below.



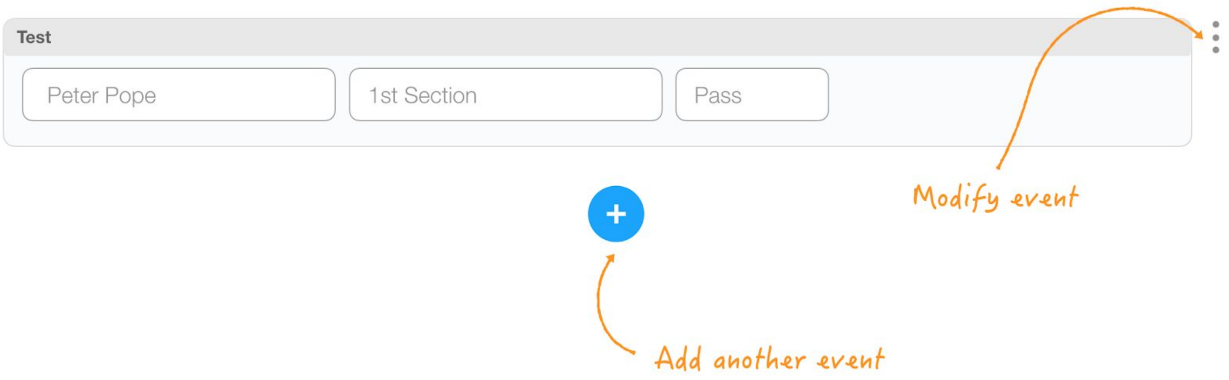
Daily Sheet - Test "Container". Instructor selects data from dropdowns to enter a student's test information.

Once any required fields are filled in the container, the Save button will activate.



Daily Sheet - Test "Container". All required fields in place - save button activates.

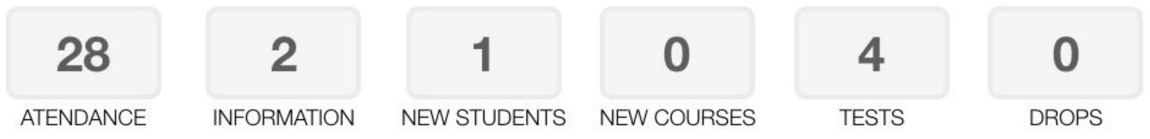
Once saved, the more dots appear, and the event can be edited or deleted. Also, a new blue plus button appears for adding additional events.



Daily Sheet - Test "Container". Saved Tests (or any event) can be edited. Once one event is saved, another can be created.

Daily Tally

As each event is added to any given day, the metrics are tallied for review. In the following example, the total attendance for the day was 28, there were 2 infos, 1 new student, and 4 students that tested.



Daily Sheet - Top Line Metrics. The attendance and events for the day are tallied.

2.2. Student Experience

In phase 1, there is no student experience. In the future, we can consider adding features. Some of the most useful may be:

- Student tracking own attendance (via a tablet interface in each school that could dovetail into the system)
- Prospective students and new students entering their own information (which would again dovetail into the system)

Both of the above feature sets would reduce instructor load on data entry, which would be good to have. However, instructors would still need to validate information such that records are well kept.

2.3. Other User Experience

More to come on data and reporting feature set, which will occur within Data Studio and shared out on a school-by-school, region-by-region, and national basis to appropriate stakeholders.

3. Data to Collect

Table Definitions

Length provided for database implementations which require length for text fields.

Informations Table

Table Name: informations

Column	**Type**	**Length**	**Description**
info_sql_id	Integer Primary Key	n/a	Index, unique
school	integer	n/a	School index from School Table
date_visited	datetime	n/a	Date of visit by Information
first_name	text	30	First name of student / instructor
last_name	text	30	Last name of student / instructor
age	integer	n/a	Student age
birth_date	datetime	n/a	Student birth date
class_group	text	10	Adult, Junior, Child
street	text	45	Street Address Line 1
street2	text	45	Street Address Line 2
city	text	25	City
state	text	5	State, Province abbreviation
postal_code	text	10	Postal (Zip) Code
country	text	25	Country
email	text	45	eMail Address
mobile_phone	text	20	Mobile Phone Number
home_phone	text	20	Home Phone Number
parental_contact	text	50	Name of Parent if Information not Adult
how_found	text	75	How the Information found the school
occupation	text	60	Occupation
interest	text	140	Interest in Martial Arts
body_issues	text	140	Injuries, Surgeries, Body Issues
notes	text	140	Notes and Comments by Instructor

Students Table

Table Name: students

Column	Type	Length	Description
student_sql_id	Integer Primary Key	n/a	Index, unique
oyd_id	integer	n/a	Oom Yung Doe ID Number, indexed
first_name	text	30	First name of student / instructor
middle_name	text	30	Middle name or initial
last_name	text	30	Last name of student / instructor
nickname	text	20	Students preferred name to be called
age	Integer	n/a	Student age
birth_date	datetime	n/a	Student birth date
school	Integer	n/a	School index from School Table
start_date	datetime	n/a	Start Date
status	text	8	Values: active, inactive, or dropped
drop_reason	text	100	Reason given for dropping
position	text	6	Intern, AI, I, … NI
rank	text	2	Student / Instructor Rank
next_rank	text	2	Student / Instructor's Next Rank
class_group	text	10	Instructor, Adult, Junior, Child
street	text	45	Street Address Line 1
street2	text	45	Street Address Line 2
city	text	25	City
state	text	5	State, Province abbreviation
postal_code	text	10	Postal (Zip) Code
country	text	25	Country
email	text	45	eMail Address
mobile_phone	text	20	Mobile Phone Number
home_phone	text	20	Home Phone Number
parental_contact	text	50	Name of Parent to Contact in Emergency
occupation	text	50	Occupation
how_found	text	75	How Student Found School e.g. Internet
intern_points	Integer	n/a	Total of points as Intern
last_test_date	datetime	n/a	Date of last test
next_test_date	datetime	n/a	Date of next test
facebook	text	100	Facebook page
instagram	text	100	Instagram page
twitter	text	100	Twitter page

The following columns are presented in this specification ofor consideration for inclusion in the Student Table:

Column	Type	Length	Description
course	text	50	Student's primary course
course_start_date	datetime	n/a	Start date of above course
coure_end_date	datetime	n/a	End date of above coruse
program	Text	50	Student's program if applicable

program_start_date	datetime	n/a	Start date of above program
program_end_date	datetime	n/a	End date of the above program
student_dedication	text	1	A, B, C – level of dedication
next_step	text	140	Student's Next Step
following_step	text	140	Student's Next Next Step
Occupation	text	50	Occupation
employer	text	50	Name of Employer
how_found	text	75	How Student Found School, e.g. Internet
training_reason	text	140	Reason for Training
short_term_goal	text	140	Short Term Goal
long_term_goal	text	140	Long Term Goal
body_issues	text	140	Injuries, Surgeries, Body Issues
notes	text	140	General Comment
picture	text	65	Link to picture file

School Validation: limit to school
Status Validation: limit to values: active, inactive, dropped
Position Validation: limit to values: Intern, AI, I, AHI, HI, ARHI, RHI, ANI, NI
Rank, Next Rank Validation: limit to values: WB, 1S, 2S, 3S, 4S, 5S, 6S, 1D, 2D, 3D, 4D, 5D, 6D, 7D
Class Group Validation: limit to values: instr, adult, junior, child
Mobile Phone Validation: limit to: numeric, -, (,)
Home Phone Validation: limit to: numeric, -, (,)
Student Dedication Validation: limit to: A, B, C

Attendance Event Table

Table name: attendance

The attendance table is an "event" table used to record an event when a student attends their daily group lesson.

Column	**Type**	**Length**	**Description**
student_sql_id	Integer Primary Key	n/a	From Student Table, index
attendance_date	datetime	n/a	Date & Time attended

MLT Attendance Event Table

Table Name: mltattendance

The testing table is an "event" table used to record an event when a student tests.

Column	**Type**	**Length**	**Description**

student_sql_id	Integer Primary Key	n/a	From Student Table, index
mlt_date	datetime	n/a	Date of test
course	Integer	n/a	Index of course taken from Course Table
lesson	Integer	n/a	Lesson number for the course

New Student Event Table

Table Name: newstudents

The testing table is an “event” table used to record an event when a student tests.

Column	**Type**	**Length**	**Description**
sql_id	Integer Primary Key	n/a	SQL id
student_sql_id	integer	n/a	From Student Table, index
oyd_id	integer	n/a	Oom Yung Doe ID Number, indexed
signup_date	datetime	n/a	Signup Date

Testing Events Table

Table Name: testingevents

The testing table is an “event” table used to record an event when a student tests.

Column	**Type**	**Length**	**Description**
sql_id	Integer Primary Key	n/a	SQL id
student_sql_id	integer	n/a	From Student Table, index
oyd_id	integer	n/a	Oom Yung Doe ID Number, indexed
test_date	datetime	n/a	Date of test
rank_tested	text	2	Two digit rank value (1S – 7D)
pass_fail	text	4	Pass or fail

Rank Tested Validation: limit to values: 1S, 2S, 3S, 4S, 5S, 6S, 1D, 2D, 3D, 4D, 5D, 6D, 7D
Pass_fail Validation: limit to values: pass, fail

Drop Events Table

Table Name: dropevents

The drop table is an “event” table used to record an event when a student drops.

Column	**Type**	**Length**	**Description**
sql_id	Integer Primary Key	n/a	SQL id

student_sql_id	integer	n/a	From Student Table, index
oyd_id	integer	n/a	Oom Yung Doe ID Number, indexed
drop_date	datetime	n/a	Date dropped
reason	text	140	Reason provided by student for dropping

Master Event Table

Table Name: masterevents

Events written to the above tables are also written to this Master Event Table.

Column	Type	Length	Description
sql_id	Integer Primary Key	n/a	index
event	text	10	info, newstudent, attendance, mlt-attendance, course, test, drop
date	datetime	n/a	Date of Event
student_sql_id	integer	n/a	Student SQL Id from Student Table if Student Event
info_sql_id	integer	n/a	Info SQL Id Info Table is Info Event
nat_area_name	text	20	Full Text name of National Area
region_name	text	25	Full Text name of Region
school_name	text	20	Full Text name of School
first_name	text	30	Student or Info’s First Name
last_name	text	30	Student or Info’s Last Name
age	integer	n/a	Student or Info’s Age
occupation	text	50	Student or Info’s Age
rank	text	2	Student’s Rank
rank_tested	text	2	Rank Student for which Tested
pass_fail	text	4	Pass or Fail Rank Tested
course_name	text	65	Full Text name of MLT Course for mlt-attendance event

~~Additional Columns~~
~~def_nat_area~~
~~def_region~~
~~def_school~~
~~first_name~~
~~last_name~~
~~--- good to have~~
~~age (from informations table or students table based on event = “informations” else “students”)~~
~~occupation (from informations table or student table, based on event)~~
~~rank (if event “information” then n/a, else from students table)~~
~~rank_tested (from testingevents)~~
~~pass_fail (from testingevents)~~
~~course (from mltattendance table)~~

User Table

Table Name: users

The user table is used for user authentication and access control.

Column	Type	Length	Description
user_id	Integer Primary Key	n/a	index
username	text	20	Username for login
hashed_password	text	100	Hash of Password
oyd_id	integer	n/a	Oom Yung Doe ID Number
access_level	Integer	n/a	System Access Level
first_name	text	30	First name of user
last_name	text	30	Last name of user
def_school	text	30	Default School
def_region	text	30	Default Region
def_nat_area	text	30	Default National Area
auth_schools	text	100	List of Authorized School Ids

Questions:

Do we need a list of authorized schools on a per user basis
Do we need a list of authorized regions on a per user basis
Do we need a list of authorized national areas on a per user basis

Access Levels are Defined as follows:

0 - Admin, granted access to everything
1 - National Instructor, Full Access to All Schools
2 - Regional Instructor, Full Access to All Schools in a Region
3 - Head Instructor, Full Access to a List of Schools
4 - Main Instructor, Full Access to a List School (usually one)
5 - Intern, Full Access to a List School (usually one)
6 - Student, Limited Access to Own Student Data (some read-only, some editable)

Student Access will be limited to Registered, Active Students in the Student Database. A separate access level will not be defined for Students as they will access the database via a separate application, and a separate validation API will be created to validate that the Student is an active registered student.

Normalization Note: first_name, last_name can be retrieved via oyd_id from student table, but are repeated here (non-normalized).

Courses Table
Table Name: courses

The Course Table provides a list of all courses which have been offered.

Column	Type	Length	Description
course_id	Integer Primary Key	n/a	index
course_name	text	65	Full Text Name of Course
course_abbrev	text	10	Abbreviation of Course Name

Schools Table
Table Name: schools

The Schools Table is a list of all schools.

Column	Type	Length	Description
school_id	Integer Primary Key	n/a	index
school_name	text	20	Full Text Name of School
main_ins_id	Integer	n/a	OYD ID of main instructor
school_region	Integer	n/a	Region Index from Region Table
street	text	45	Street Address Line 1
street2	text	45	Street Address Line 2
city	text	25	City
state	text	5	State, Province abbreviation
postal_code	text	10	Postal (Zip) Code
country	text	25	Country
email	text	45	eMail Address
school_phone	text	20	School Phone Number
status	integer	1	(O)pen, (C)losed
standing	integer	1	School Standing – (G)ood, (B)ehind

Status Validation: limit to O, C
Standing Validation: limit to G, B

Regions Table
Table Name: region

The Regions Table is a list of all regions.

Column	Type	Length	Description
region_id	Integer Primary Key	n/a	index

region_name	text	25	Full Text Name of Region
region_abbrev	text	10	Abbreviation of Region Name
main_reg_id	Integer	n/a	OYD ID of head regional instructor
reg_team_ids	text	45	OYD IDs of regional team members
nat_area	Integer	n/a	National area Index from National Area Table
street	text	45	Street Address Line 1
street2	text	45	Street Address Line 2
city	text	25	City
state	text	5	State, Province abbreviation
postal_code	text	10	Postal (Zip) Code
country	text	25	Country
email	text	45	eMail Address
phone	text	20	School Phone Number
status	integer	1	(O)pen, (C)losed
standing	integer	1	Region Standing – (G)ood, (B)ehind

Status Validation: limit to O, C
Standing Validation: limit to G, B

National Area Table - ‘Areas’
Table Name: areas

The National Area Table, ‘Areas’, is a list of all national areas.

Column	**Type**	**Length**	**Description**
nat_area_id	Integer Primary Key	n/a	index
area_name	text	20	Full Text Name of National Area
area_abbrev	Text	10	Abbreviation of National Area name

Object Model and API (Application Program Interface)

Introduction
The backbone of the Daily Information Application is tracking events so that these events can be fed to Google Data Studio for data analytics and visualization.

Actions on objects cause the events to “fire” writing those events to the table associated with the event as well as to the Master Event table.

Events Handled through Objects
The events that are captured through actions on objects are:

- New Information
- New Student
- New Course
- Test
- Student Dropped

Events Handled through Functions

The events that are captures via a function call due to the nature of the event include:

- Daily Attendance
- MLT Attendance

Database Abstraction Model

The Database is abstracted by an object model which models both an individual row of a table, e.g. a Student, as well as the associated table of the database e.g. the Student Table, each as objects.

For example:

> The Student object allows the developer to add a new student to the database, retrieve an existing student from the database for editing, or to update an edited student to the database.
>
> The Student Table object allows for querying multiple students based on a parameters passed to the query method thus abstracting the database query language into a parameterized method call.
>
> Given that events must be captured, the action of adding a new student to the database through the Student Object, also “fires” an event which gets written to the New Student Event Table as well as the Master Event table.

Student Object

The student object represents a single student and provides the ability to retrieve (get) an existing student from the database, write (put) a new student to the database, or update an existing student in the database.

The student object does not support delete as students who drop continue to be tracked, remain in the database, and their status is changed to “dropped”

Attributes

Usage: student.attr[‘*attribute name*’]

Attributes for the Student Object are stored in a Dictionary.

```
self._sql_id = sql_id                        # internal use only, do not change
self.attrs = {'student_sql_id': None,   # internal use only, do not change
       'oyd_id': oyd_id,
       'first_name': first_name,
       'middle_name': middle_name,
       'last_name': last_name,
       'nickname': nickname,
       'age': age,
       'birth_date': birth_date,
       'school': school,
       'start_date': start_date,
       'status': status,
       'position': position,
       'rank': rank,
       'next_rank': next_rank,
       'class_group': class_group,
       'street': street,
       'street2': street2,
       'city': city,
       'state': state,
       'postal_code': postal_code,
       'country': country,
       'email': email,
       'mobile_phone': mobile_phone,
       'home_phone': home_phone,
       'parental_contact': parental_contact,
       'intern_points': intern_points,
       'last_test_date': last_test_date,
       'next_test_date': next_test_date,
       'facebook': facebook,
       'instagram': instagram,
       'twitter': twitter
       }

 self.select_status = {
       "All":None, "Active":"active", "Inactive":"inactive",
       "Dropped":"dropped"}

   self.select_position = {
       "None":"None", "AI":"AI", "I":"Instr", "AHI":"AHI", "HI":"HI",
       "ARHI":"ARHI", "RHI":"RHI", "ANI":"ANI", "NI":"NI"}

   self.select_rank = {"WB":"WB", "1S":"1S", "2S":"2S", "3S":"3S",
       "4S":"4S", "5S":"5S", "6S":"6S", "1D":"1D", "2D":"2D",
       "3D":"3D", "4D":"4D", "5D":"5D", "6D":"6D", "7D":"7D"}
```

```
self.select_next_rank = {"1S":"1S", "2S":"2S", "3S":"3S", "4S":"4S",
   "5S":"5S", "6S":"6S", "1D":"1D", "2D":"2D", "3D":"3D",
   "4D":"4D", "5D":"5D", "6D":"6D", "7D":"7D", "8D":"8D"}

self.select_class_group = {"instr":"Instr", "adul":"Adult", "junior":"Junior",
   "child":"Child"}

# Matching dictionary of Human Reacable Titles for Student Atributes
self.labels = {'student_sql_id': 'SQL ID',
   'oyd_id': 'OYD ID:',
   'first_name': 'First Name:',
   'middle_name': 'Middle Name:',
   'last_name': 'Last Name:',
   'nickname': 'Nickname:',
   'age': 'Age:',
   'birth_date': 'Birth Date:',
   'school': 'School:',
   'start_date': 'Start Date:',
   'status': 'Status:',
   'position': 'Position:',
   'rank': 'Rank:',
   'next_rank': 'Next Rank:',
   'class_group': 'Class:',
   'street': 'Street:',
   'street2': 'Street2:',
   'city': 'City:',
   'state': 'State / Prov:',
   'postal_code': 'Postal Code:',
   'country': 'Country:',
   'email': 'eMail:',
   'mobile_phone': 'Mobile Phone:',
   'home_phone': 'Home Phone:',
   'parental_contact': 'Parental Contact:',
   'intern_points': 'Intern Points:',
   'last_test_date': 'Last Test Date:',
   'next_test_date': 'Next Test Date:',
   'facebook': 'Facebook:',
   'instagram': 'Instagram',
   'twitter': 'Twitter:'
   }

# Matching dictionary of UI input types for Student Attributes
self.label_types = {'student_sql_id': "hidden",
   'oyd_id': 'number',
   'first_name': "text",
'middle_name': "text",
'last_name': "text",
'nickname': "text",
'age': "number",
'birth_date': 'date',
'school': 'number',        # select
'start_date': 'date',
'status': 'text',          # select
'position': 'text',        # select
'rank': 'text',            # select
'next_rank': 'text',       # select
'class_group': 'text',     # select
'street': 'text',
'street2': 'text',
'city': 'text',
'state': 'text',           # select
'postal': 'text',
'country': 'text',         # select
'email': 'email',
'mobile_phone': 'text',
'home_phone': 'text',
'parental_contact': 'text',
'intern_points': 'number',
'last_test_date': 'date',
'next_test_date': 'date',
'facebook': 'text',
'instagram': 'text',
'twitter': 'text'
}
```

Methods
The student object provides the following methods.

Init Method
An instance of the student object is initialized with all attributes defaulted to None, Zero or the equivalent.

Put Method
Usage: student.put(db)
Parameters:

    db - database object,

Saves a new Student Object as a new student (new row) in the Database.

The Put Method writes an event into the New Student Event table and the Master Event table.

Returns a tuple (return code, error message)
    Return Code: 0 for success, 1 for error
    Message: System generated error message

Get Method

Usage: student.get (db)
Parameters:
    db - database object,

Populates an instantiated Student Object from the Database based on the student_sql_id attribute.

Returns a tuple (return code, error message)
    Return Code: 0 for success, 1 for error
    Message: System generated error message

Update

Used to update multiple attributes at the same time.
Please do not use this method to update:
    status
    last_test_date
Please use Update_attrs to update each of these attributes so that changes to these attributes can be captured as events.

Usage: student.update(db)
Parameters:
    db - database object,

Updates the Student Object to the Database.

Returns a tuple (return code, error message)
    Return Code: 0 for success, 1 for error
    Message: System generated error message

Update_attr

Used to update a single attribute to the database
Please use this method to update status, next_test_date so that that events may be captured.

Usage: student.update_attr(db, label)
Parameters:
    db - database object,

label - name of single attribute that was updated
Updates the specified attribute in the Student Object to the Database.

The update_attr method checks for attribute changes and writes the associated event to the proper event table as well as to the Master Event table capturing those event. The attributes that are tracked for changes are:

- student.status
- Last_test_date

Note: most events are captured when the Student object attributes are updated using Student().update_attrs. However, it can only catch a successful test. All failed tests must be written directly using the Testing Event Object and the Master Event Object.

Returns a tuple (return code, error message)
Return Code: 0 for success, 1 for error
Message: System generated error message

Student Table Object

The student table object provides methods for querying lists of students from the database. For example, a list of students for a single school can be queried so that they can be presented for marking attendance.

Attributes

students = []
status = None
school = None
region = None
nat_area = None
offset = None
limit = None

Methods

Init

An instance of the student table object is initialized with all attributes defaulted to None, Zero or the equivalent.

QueryList Method

Usage: studentTable.queryList (status = None, school = None, region = None, nat_area = None, offset = None, limit = None)

Provides a list of students based on the parameters passed to the method.
Supports pagination in the UI through offset, the starting row to return and limit, the number of rows to returns.

Parameters

status – acceptable values are: active, inactive, or dropped, default is none which returns all
school – id of the school, default is None which returns all schools
region – id of the region, default is None which returns all regions
nat_area – id of the national area, default is None which returns all national areas
offset – starting row to return, default is None which start the return at the first row of data
limit – number of rows to return, default is None which returns all rows

Updates the students List attribute appending one tuple for each row in the query returned.

Returns a tuple (return code, error message)
    Return Code: 0 for success, 1 for error
    Message: System generated error message

Information Object

The information table does not support delete as the data is tracked over time.

Attributes

Usage: information.attr[‘*attribute name*’]

Attributes for the Student Object are stored in a Dictionary.

```
_sql_id = sql_id                          # internal use only, do not change
attrs = {'info_sql_id': info_sql_id,
        'school': school,
        'first_name': first_name,
        'last_name': last_name,
        'age': age,
        'birth_date': birth_date,
        ‘Class_group’: class_group,
        'date': date,
        'street': street,
        'street2': street2,
        'city': city,
        'state': state,
        'postal': postal,
        'country': country,
        'email': email,
        'mobile_phone': mobile_phone,
        'home_phone': home_phone,
        ‘how_found’: how_found,
        ‘occupation’: occupation,
        'interest': interest,
        'body_issues': body_issues,
        'notes': notes
        }
```

Methods

Init Method

An instance of the student object is initialized with all attributes defaulted to None, Zero or the equivalent.

Put Method

Usage: information.put(db)

Saves a new Information Object as a new information (new row) in the Database.

The Put Method writes an event into the New Information Event table and the Master Event table.

Returns a tuple (return code, error message)
    Return Code: 0 for success, 1 for error
    Message: System generated error message

Get Method

Usage: information.get(db)

Populates an instantiated Information Object from the Database based on the info_sql_id attribute.

Returns a tuple (return code, error message)
    Return Code: 0 for success, 1 for error
    Message: System generated error message

Update

Usage: information.update(db)
Parameters:
    db - database object,

Updates the Information Object to the Database.

Returns a tuple (return code, error message)
    Return Code: 0 for success, 1 for error
    Message: System generated error message

Update_attr
Usage: information.update_attr(db, label)
Parameters:
    db - database object,
    label - name of single attribute that was updated
Updates the specified attribute in the InformationObject to the Database.

Returns a tuple (return code, error message)
    Return Code: 0 for success, 1 for error
    Message: System generated error message

School Object
The school table does not support delete as the school_id index is used in other tables which may be long lived.

Attributes
```
attrs = {'school_id': None,   # internal use only, do not change
          'school_name': school_name,
          'main_ins_id': main_ins_id,
          'school_region': school_region,
          'street': street,
          'street2': street2,
          'city': city,
          'state': state,
          'postal_code': postal_code,
          'country': country,
          'email': email,
          'school_phone': school_phone,
          'status': status,
          'standing': standing
          }

select_status = {'Open':'0', 'Closed':'1'}
```
select_status = {'Open':'0', 'Closed':'1'}

Methods
Get Method
Usage: school.get(db)

Populates an instantiated School Object from the Database based on the school_id attribute.

Returns a tuple (return code, error message)
    Return Code: 0 for success, 1 for error

Message: System generated error message

Put Method

Usage: school.put(db)

The Put Method writes the new school object to the schools table.

Returns a tuple (return code, error message)
- Return Codes:
    - 0 for success
    - 1 for error: school already exists
    - 2 for error: failed to add new school
- Message: System generated error message

Update Method

Usage: school.update(db)
Parameters:
- db - database object,

Updates the School Object to the Database.

Returns a tuple (return code, error message)
- Return Code: 0 for success, 1 for error
- Message: System generated error message

Schools Table Object

Attributes

```
self.region = None
self.schools = []
self.limit = None
self.offset = None
```

Methods

Count Method

Usage: schools_table.count(db, [region])

Returns a count of the number of Schools in the schools table with the Region field equal to region. The region parameter is optional. If region is not included, a count of all schools is returned.

Query_All Method
Usage: schools_table.query_all (db)

Populates schools_table.schools List with a list of all schools in the schools table.

Return Code: 0 for success, 1 for error

Query Method
Usage: schools_table.query_all (db, [region])

Populates schools_table.schools List with a list of all schools in the schools table with a Region field equal to region. The region parameter is optional. If region is not included, all schools are returned.

Return Code: 0 for success, 1 for error

QueryRange Method
Usage: schools_table.query_all (db, limit, offset, [region])

Populates schools_table.schools List with a list of all schools starting at “limit” returning an “offset” number of schools where the Region field equals region. region is optional.

Return Code: 0 for success, 1 for error

Region Object
The region table does not support delete as the region_id index is used in other tables which may be long lived.

Attributes
```
attrs = {'region_id': None,   # internal use only, do not change
         'region_name': region_name,
         'nat_area': nat_area,
         'reg_head_id': reg_ins_id,
         'reg_team_names': reg_team_names,
         'phone': phone,
         'email': email,
         'status': status,
         'standing': standing
         }
```

Methods

Get Method

Usage: region.get(db)

Populates an instantiated Region Object from the Database based on the sql_id attribute.

Returns a tuple (return code, error message)
- Return Codes:
  - 0 for success
  - 1 for error
- Message: System generated error message

Put Method

Usage: region.put(db)

Saves a new Region Object as a region (new row) in the Database.

The Put Method writes the new region object to the regions table.

Returns a tuple (return code, error message)
- Return Codes:
  - 0 for success
  - 1 for error: region already exists
- Message: System generated error message

Update Method

Usage: region.update(db)

Parameters:
- db - database object,

Updates the Region Object to the Database.

Returns a tuple (return code, error message)
- Return Code: 0 for success, 1 for error
- Message: System generated error message

Regions Table Object

Attributes

```
self.regions = []
self.limit = None
self.offset = None
```

Methods

Count Method

Usage: regions_table.count(db)

Returns a count of the number of Regions in the regions table.

Query_All Method

Usage: regions_table.query_all (db)

Populates regions_table.regions List with a list of all regions in the regions table.

Return Code: 0 for success, 1 for error

QueryRange Method

Usage: regions_table.query_all (db, limit, offset)

Populates regions_table.regions List with a list of all regions starting at “limit” returning an “offset” number of regions.

Return Code: 0 for success, 1 for error

National Area Object

The national area (areas) table does not support delete as the nat_area_id index is used in other tables which may be long lived.

Attributes

```
attrs = {'nat_area_id': None,   # internal use only, do not change
        'area_name': area_name,
        'area_abbrev': area_abbrev
        }
```

Methods

Get Method

Usage: natarea.get(db)

Populates an instantiated NatArea Object from the Database based on the nat_area_id attribute.

Returns a tuple (return code, error message)

    Return Code: 0 for success, 1 for error
    Message: System generated error message

Put Method
Usage: natarea.put(db)

The Put Method writes the new NatArea object to the areas table.

Returns a tuple (return code, error message)
  Return Code: 0 for success, 1 for error
  Message: System generated error message

Update Method
Usage: natarea.update(db)
Parameters:
  db - database object,

Updates the NatArea Object to the Database.

Returns a tuple (return code, error message)
  Return Code: 0 for success, 1 for error
  Message: System generated error message

National Area Table Object

Attributes

```
self.areas = []
self.limit = None
self.offset = None
```

Methods

Count Method
Usage: natareas_table.count(db)

Returns a count of the number of Regions in the regions table.

Query_All Method
Usage: natareas_table.query_all (db)

Populates natareas_table.areas List with a list of all areas in the areas table.

Return Code: 0 for success, 1 for error

QueryRange Method
Usage: natareas_table.query_all (db, limit, offset)

Populates natareas_table.areas List with a list of all areas starting at “limit” returning an “offset” number of areas.

Return Code: 0 for success, 1 for error

Course Object

The courses table does not support delete as the course_id index is used in other tables which may be long lived.

Attributes

```
attrs = {'course_id': None,
         'course_name': name,
         'course_abbrev': course_abbrev
         }
```

Methods

Get Method

Usage: course.get(db)

Populates an instantiated Course Object from the Database based on the course_id attribute.

Returns a tuple (return code, error message)
    Return Code: 0 for success, 1 for error
    Message: System generated error message

Put Method

Usage: course.put(db)

The Put Method writes the new Course Object to the courses table.

Returns a tuple (return code, error message)
    Return Code: 0 for success, 1 for error
    Message: System generated error message

Update Method

Usage: course.update(db)
Parameters:
    db - database object,

Updates the Course Object to the Database.

Returns a tuple (return code, error message)
    Return Code: 0 for success, 1 for error
    Message: System generated error message

Courses Table Object

Attributes

```
self.courses = []
self.limit = None
self.offset = None
```

Methods

Count Method

Usage: courses_table.count(db)

Returns a count of the number of courses in the courses table.

Query_All Method

Usage: courses_table.query_all (db)

Populates courses_table.areas List with a list of all courses in the courses table.

Return Code: 0 for success, 1 for error

QueryRange Method

Usage: courses_table.query_all (db, limit, offset)

Populates courses_table.courses List with a list of all courses starting at "limit" returning an "offset" number of courses

Return Code: 0 for success, 1 for error

User Object

Note: it is the recommendation of HI Tom, that some of the methods on this object, be moved to a separate admin package found in admin.py in the admin directory.

Attributes

```
attrs = {'user_id': None,   # internal use only, do not change
         'username': None,
         'hashed_password': None,        # not set on validation
         'oyd_id': None,
         'access_level': None,
         'first_name': None,
         'last_name': None,
         'def_school': None,
         'def_region': None,
         'def_nat-area': None,
         'school_list': None,
         'region_list': None
```

Methods

Authenticate Method

Usage: user.authenticate(db, username, password)

Authenticates the user, defined by username, and clear text password, password. Checks the Hash of the password for the user, username.

Returns True is Authenticate, False if Authentication fails

Get Method

Usage: user.get(db)

Gets the User data, using the attr['user_id'] to query, from the users table.

Returns 0 for success, 1 for failure

Put Method

Suggested that this be moved to admin.py

Usage: user.put(db, password)

Adds the user to the users table with a hashed password of the password parameter.

Returns a tuple (return code, error message)

  Return Code: 0 for success, 1 for error
  Message: System generated error message

Update Method

Suggested that this be moved to admin.py and replaced by a Method that only allows for updating the password.

Usage: user.update(db, [password])

Updates the user to the users table of the database. If the password parameter is included, a the new password is hashed and stored.

Returns 0 for success, 1 for failure

Delete Method
Suggested that this be moved to admin.py
Usage: user.delete(db)

Deletes the user, defined by attr[‘user_id’], from the users table.

Returns 0 for success, 1 for failure

User Table Object
The user table object is found in admin.py as part of the admin package found in the admin directory.

Attributes
```
        self.user = User()
```

Methods
Count Method
Usage: count_rows(db)

Returns the number of users in the users tables, or None.

Get_All Method
Usage: get_all (db, limit, offset)

Returns users, as a row of data, from “limit” for an “offset” number of users or the end of the table.

Returns a tuple (return code, error message)

Return Code: 0 for success, 1 for Error: “No Users Found in Database”,
2 for Error: Query for Users Failed”

Rows - A Python List of Rows - each Row a user in the users table

Message: System generated error message

Master Event Object
Attributes
```
        self.attrs = {'sql_id': None,
                      'event': event,
                      'date': date,
                      'student_sql_id': student_sql_id,
                      'info_sql_id': info_sql_id,
                      'nat_area_name': nat_area_name,
                      'region_name': region_name,
                      'school_name': school_name,
                      'first_name': first_name,
                      'last_name': last_name,
                      'age': age,
                      'occupation': occupation,
                      'rank': rank,
                      'rank_tested': rank,
                      'pass_fail': pass_fail,
                      'course_name': course_name
                      }
```

Methods

Put Method

Usage: MasterEvent().put (db)

Writes the event, defined by the attributes of the Master Event object, to the Master Events table - masterevents.

Note: this method is called by Drop Event, Testing Event, New Student Event, Attendance, and MLTAttendance automatically.

Returns 0 for success, 1 for error.

New Students Event Object

Attributes

```
          self.attrs = {'sql_id': None,
                          'student_sql_id': student_sql_id,
                          'oyd_id': oyd_id,
                          'signup_date': signup_date
                          }
```

Methods

Put Method

Usage: New_Students_Event().put (db)

Writes the event, defined by the attributes of the New Student Event object, to the New Student Events table - newstudents.

~~Calls MasterEvent.put() to write the event to the Master Events table.~~

Returns 0 for success, 1 for error.

Testing Event Object

Note: most events are captured when the Student object attributes are updated using Student().update_attrs. However, it can only catch a successful test. All failed tests must be written directly using the Testing Event Object and the Master Event Object.

Attributes

```
        self.attrs = {'sql_id': None,
                      'student_sql_id': student_sql_id,
                      'oyd_id': oyd_id,
                      'test_date': test_date,
                      'rank_tested': rank_tested,
                      'pass_fail': pass_fail
                      }
```

Methods

Put Method

Usage: Testing_Event().put (db)

Writes the event, defined by the attributes of the Testing Event object, to the Testing Events table - testingevents.

~~Calls MasterEvent.put() to write the event to the Master Events table.~~

Returns 0 for success, 1 for error.

Drop Event Object

Attributes

```
        self.attrs = {'sql_id': None,
                      'student_sql_id': student_sql_id,
                      'oyd_id': oyd_id,
                      'drop_date': drop_date,
                      'reason': reason
                      }
```

Methods

Put Method

Usage: Drop_Event().put (db)

Writes the event, defined by the attributes of the Drop Event object, to the Drop Events table - dropevents.

Calls MasterEvent.put() to write the event to the Master Events table.

Returns 0 for success, 1 for error.

Daily Attendance Function

Usage: attendanceDaily (db, student_sql_id_list, date, class_group)

Writes multiple events, based on a list of students who attended a single class, into the Attendance event table.

Parameters:

    db - database object
    student_sql_id_list – List, list of ids that attended
    date – Date, date of attendance
    class_group – Text, values limited to: instructor, adult, junior, child

For each id in the List, writes a row into the Attendance Table with the date and class_group specified.

Calls MasterEvent.put() to write the event to the Master Events table.

MLT Attendance Function

Usage: attendanceMLT (db, student_sql_id, date, course, lesson)

Writes a single event for one (1) student into the MLT Attendance event table.

Parameters:

    db - database object
    student_sql_id – Integer, student's sql id
    date – Date, date of attendance
    course – Integer, from course table
    lesson – Integer

For each id in the List, writes a row into the Attendance Table with date and class_group specified.

Calls MasterEvent.put() to write the event to the Master Events table.

4. Data / Reporting to Display

More coming soon...

5. Phase 2

This section details ideas and/or features slated for Phase 2 of the Daily Program.

- Automatic notification of Daily Information to Slack. Information including without limitation:
    - Number of Informations today
    - Number of New Students today
    - Number of New Courses set today
    - Number of Tests completed today (by rank?)
    - Number of Drops today