CSAL Database Project Introduction

Craig Kelly

November 14, 2014

Contents

1 Introduction

This repository contains code for storing and displaying data stored as part of the CSAL project. The use cases and the code produced are described below. The short version is that the data is stored in a MongoDB instance, there is a C# library for accessing the database, and there is a Web API wrapping the DLL. In addition, the Web API server provides a very simple ASP.NET MVC application for viewing the data. Because of the dependence of the Web API on the "core" library, the only deployment information is below in 6.5 Deployment

2 Use Cases

2.1 Direct Logging

Services running on the same server as the database (i.e. ACE) want the ability to write JSON data log entries without POST'ing to an HTTP endpoint. Note that while the architecture of the project allows this use case (and it is currently used in production), a safe alternative would be to force all applications to write via the ReST API (see 2.2 Logging via ReST Endpoint).

2.2 Logging via ReST Endpoint

Applications may post a JSON record representing an ACE turn to a public endpoint for persisting in the database. Note that this is the recommended way to save data to the database (contrast with 2.1 Direct Logging)

2.3 Teacher Status Check

Teachers need to be able to see students' progress in the lesson. There should be a way to see how the entire class is doing, how the class is doing on a lesson, how a student is doing across the lessons, and how a specified student is doing on a specific lesson. Note that if this application grows, this Use Case should be broken out into small chunks

2.4 Data Administration

There must be a way for administrators (not teachers) to initialize and edit the database. Specifically, class, lesson, and student data should be configured for expected logging (via either use case 2.1 Direct Logging or 2.2 Logging via ReST Endpoint).

3 A Word About Dates

Any system logging or reporting events that occur in time must deal with the how time is represented. It is important to remember that this application will generally deal with dates with the standard .NET libraries, but the actual dates are stored in a MonogDB database. In addition, for some tasks it is better to have a quick way of determining event order and spacing; as a result, some dates are stored as a number of milliseconds since a specified "epoch".

3.1 General Date Handling

Generally speaking, dates are handled as instances of .NET DateTime objects. Local time and formats are used, so as of now dates are displayed using the US standard and in Central Standard Time ("Memphis" or "Chicago" time).

However, MongoDB prefers to store dates in UTC (or "Zulu" time). As a result, applications using a DateTime instance that has been read back from the database will need to convert the DateTime instance via a call to `DateTime.ToLocalTime`. Currently this only applies to DateTime-valued properties on classes in the namespace `CSALMongo.Model`.

3.2 Epoch-Based Dates

There are two float-valued time-related fields in a Turn as stored in the Student Acts collection in the database (see 4.4 Student Actions).

The first is "duration". It comes from the logging application and represents the length of the turn being logged in milliseconds.

The second is "DBTimestamp". It is generated by the logging acceptance code in our class `CSALMongo.CSALDatabase`. It is calculated as the number of milliseconds since January 1st of the year `CSALMongo.TurnModel.ConvLog.EPOCH_YR`, which is currently 2010.

If "DBTimestamp" is found in the JSON data passed in (as if the logging application generated it), it is treated as the current time. This allows previously saved turns to be "re-posted" to the database via the ReST API. For instance, the script scripts/turn_copy.py in this project relies on this functionality.

3.3 Date Correction

Some instances of "DBTimestamp" might be flawed; for instance, records collected before this functionality were added have spurious, too-close timestamps. To attempt to keep the time intervals for the test system (or a production system with some kind of issues) displayed in a sane manner, we use the method `CSALMongo.Model.StudentLessonActs.FixupTimestamps`. This method insures that the timestamp of any turn n is a "minimally pretty valid" timestamp.

Every turn's timestamp is fixed so that $ts_n \geq ts_{n-1} + dur_{n-1} + 200$ where "ts" is a timestamp and "dur" is a duration. This insures that a Turn won't appear to happen during the previous turn. Also note the arbitrarily chosen extra margin of 200 milliseconds.

4 Database

The main documentation for the JSON logging record is available in a Google doc. Please see the documents "CSAL Data" at `https://docs.google.com/document/d/19nJZMReWpTat_tjNeOvA7oa8rDD0KmQ5oKaHnn2HmwE` and "AutoTutor Conversation Engine (ACE) Web API (CORS Version)" at `https://docs.google.com/document/d/1ZRlj7e5u4PQSlggCD--yZ5EsB2KCZzt7P8MEI6HB9So`

The various database entities are described below. The JSON logging data described above is stored in the entity described in 4.4 Student Actions . You may also see how the C# classes for this data (both the JSON logging format and the database entities below) are structured by looking in the CSALMongo project or the CSALMongo.chm compiled documentation in this directory.

The server is autotutor.x-in-y.com and the MongoDB database should be named csaldata. There are four collections: classes, lessons, students, and studentActions.

4.1 Class

There is one document in this collection per class. It contains a list of the students in the class and the lessons used. Please see 4.4 Student Actions for details on auto-creation and updating.

4.2 Student

There is one document in this collection per student. Please see 4.4 Student Actions for details on auto-creation and updating.

4.3 Lesson

There is one document in this collection per lesson. Please see 4.4 Student Actions for details on auto-creation and updating.

4.4 Student Actions

There is one document in this collection per student per lesson. Any time a JSON logging record is saved for a student in a lesson, it is appended to the Turns list in the corresponding document in this collection. Although it is preferred to have the class, lesson, and student documents matching this information pre-populated, a minimal version of each entity will be created if it is not present when the data is logged. Auto-created entities will have a property named AutoCreated set to true.

To help with queries, we also update the class, student, and lesson documents when we receive turn data like so:

- 4.1 Class - update the fields Students and Lessons
- 4.3 Lesson - update the fields LastTurnTime, Students, AttemptTimes, StudentsAttempted, StudentsCompleted, and URLs
- 4.2 Student - update the fields LastTurnTime and TurnCount

5 CSALMongo DLL

The "base" or "core" DLL contains the model classes for JSON logging format, the model classes for the database, the actual database interface class, and some supporting code. The project is documented via XML documentation which has been compiled into the CSALMongo.chm in this directory.

There is also a unit test project named CSALMongoUnitTest. It uses the Unit Testing facilities availble with Visual Studio 2013. The tests are broken into five categories of testing:

1. Model Testing for methods added the model classes for parsing, information, etc.
2. Database Operations Testing for actual database operations exposed by the main class

3. Database Utility Testing for helper or utility methods exposed by the main database class
4. Utility Testing for helper or utility functions **outside** the main database class.

6 CSALMongo Web API

6.1 ReST API

The ReST API is exposed via a .NET Web Api project (that also houses a GUI - see 6.2 User Interface).

Since this is a ReST API, there is a URL namespace complete with expected verbs and payloads. They are documented below. You may also see the Python script `../scripts/db_init.py` for an example of using ReST API. Essentially, the rules are:

- Lesson, Class, and Student ID's placed in a URL should be double-escaped (see also RenderHelp in the Web API project)
- When POST'ing, set the header "Content-Type" to "application/json"

Here are the API endpoints. It should be assumed that for local workstation testing the url would begin with `http://localhost:62702`. For the production URL, the proper prefix would be `http://autotutor.x-in-y.com/csaldb`.

- GET /api/classes - returns all Classes in the database
- GET /api/classes/$id (where $id is ClassID) - returns the specified Class
- POST /api/classes/$id (where $id is ClassID) - the posted body should be a JSON document matching Model.Class.
- GET /api/lessons - returns all Lessons in the database
- GET /api/lessons/$id (where $id is LessonID) - return the specified Lesson
- POST /api/lessons/$id (where $id is LessonID) - the posted body should be a JSON document matching Model.Lesson.
- GET /api/studentreading/$id (where $id is StudentID) - return list of reading URL's and timestamps (in order they were POST'ed)
- POST /api/studentreading - the posted body should be a JSON document with two fields: UserID and TargetURL. UserID is a StudentID.
- GET /api/studentsatlocation/$location (where location is a string matching the location field of one or more classes). Return a list of students (JSON formatted to a match Model.Student) that are in classes matching the specified location.
- GET /api/students - returns all Students in the database
- GET /api/students/$id (where $id is StudentID) - return the specified Student
- POST /api/students/$id (where $id is StudentID) - the posted body should be a JSON document matching Model.Student.

- GET /api/turn/$lesson/$user (where $lesson is a LessonID and $user is a Student/Subject ID) - return a list of logged turns in their original JSON format
- POST /api/turn - the posted JSON body should match the format as described in the documents linked in 4 Database or in the TurnModel.ConvLog.
- GET /api/turnrollup/$lesson/$user (where $lesson is a LessonID and $user is a Student/Subject ID) - return a "rolled up" list of turns. Each item in the list represents an attempt as logged as part of the JSON API.
- GET/POST /api/maker - for testing and should probably not be used at this point

6.2 User Interface

The User Interface is an ASP.NET MVC web application, served by the Home controller class, rendered via Razor. It uses jQuery and Bootstrap for UI automation and styling. Various jQuery UI plugins are also used (notably DataTables and Sparklines).

The application maintains a URL namespace similar to the ReST API, but under the Home directory. As above, It should be assumed that for local workstation testing the url would begin with `http://localhost:62702`. For the production URL, the proper prefix would be `http://autotutor.x-in-y.com/csaldb`.

Note that if a user access one of these endpoints (with the exception of those related to authentication) and is not logged in, a login/redirection will occur. In addition, a users' views will be filtered by classes that their login email matches (in the TeacherName field). The one exception is administrators. Please see 6.3 Authentication for details.

- /home/logout - clears the current user session and redirects to the index page
- /home/login - starts the Google OAuth2 login process
- /home/oauth2callback - the endpoint the Google OAuth2 servers will use for redirection after user authentication
- /home/index - the default/index page
- /home/testing - shows the silly testing page, which should probably be removed
- /home/classes - shows the list of classes viewable by the current user
- /home/classdetails/$id (where $id is a ClassID) - display details for the indicated class
- /home/studentlessondrill/$lesson/$user (where $lesson is a LessonID and $user is a Student/Subject ID). Construct and display a drill-down into a student's work on a single lesson
- /home/studentlessondevselect (for administrators only) - allow the user to select a student/lesson combination for viewing a debug drill-down
- /home/studentlessondevview/$lesson/$user (where $lesson is a LessonID and $user is a Student/Subject ID - for administrators only). Construct and display a debug drill-down into a student's work on a single lesson
- /home/lessons - shows the list of lessons viewable by the current user
- /home/lessondetails/$id (where $id is a LessonID) - display details for the indicated lesson
- /home/students - shows the list of students viewable by the current user

- /home/studentdetails/$id (where $id is a StudentID) - display details for the indicated student
- /home/materials - view the current hard-coded list of teaching materials

6.3 Authentication

Authentication is handled via Google OAuth2. Administrators are identified by email address in the web.config file. Teachers are given access to class information if the address from their OAuth2 profile matches the teacher name for the class.

6.4 Logging

This document generally refers to logging to identify ACE turn records sent in JSON format and stored in the MongoDB collection studentActions. However, the Web API and GUI must log records as well. Currently this is fairly simple; in addition to default IIS logging, the Elmah library is used to log unhandled exceptions and any errors displayed via the custom user error page. The Elmah log is only stored in-memory (and so is transient) and can be accessed at the main URL at `http://autotutor-x-in-y.com/csaldb/elmah`.

6.5 Deployment

The entire application should be deployed via Visual Studio packaging and IIS application import. Please note that the application is already deployed to its own App Pool on the production server, so a new deployment should be an overwrite (not a new application).

Any deployment should also be accompanied by an annotated tag in the Git repository.

7 Sequence Diagrams

7.1 Example of User Interaction