Student Lap Tracker

Detailed Design Documentation

Patrick Oyarzun, Joe DeHart
Vaishali Patel, Duke Meche, Robert Knott

ABSTRACT
This document describes the design decisions and processes used for implementing the Student Lap Tracker project. Of the static models being used, the class diagram shows the interaction between classes and the data persistence model shows the entities and the data that they contain. Of the dynamic models being used, the import and export sequences are described, which make use of the python-excel library.

TABLE OF CONTENTS

Abstraction	2
Introduction	3
GUI (Graphical User Interface) Design - Home Page - Login Page - Import Page - Export Page - Tracking and Scan Page - Milestone Config	3-6
Static Model - Class Diagram - Data Persistence Model	7-9
Dynamic Model - Import Sequence - Export Sequence	9-11
Rationale For Your Detailed Design Model	12
Traceability From Requirements to Detailed Design Model	12
Evidence the Design Model has Been Placed Under Configuration Management	13
References	13

LIST OF FIGURES (3, 6, 7)

Figure 1. Class Diagram	3
Figure 2. Import Sequence	7
Figure 3. Export Sequence	8

LIST OF TABLES

Table 1. Teacher Entity 3-4

Table 2. Student Entity 4

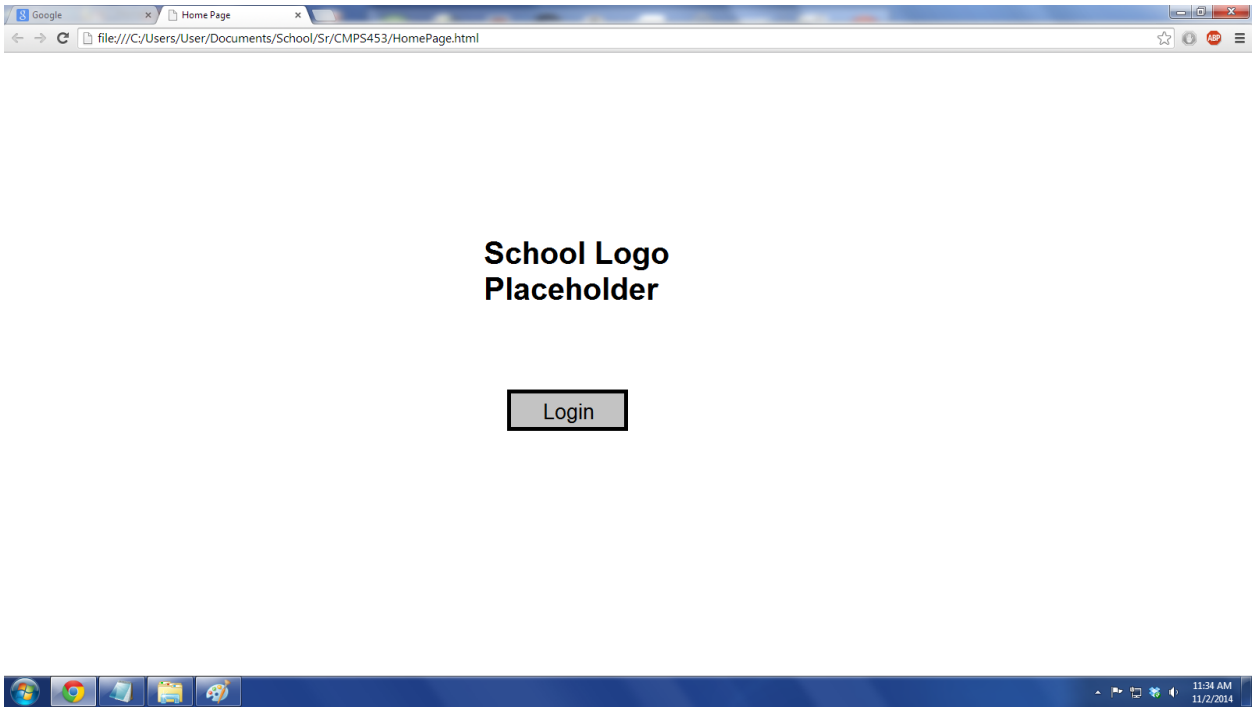
Table 3. Lap Entity 4-5

INTRODUCTION

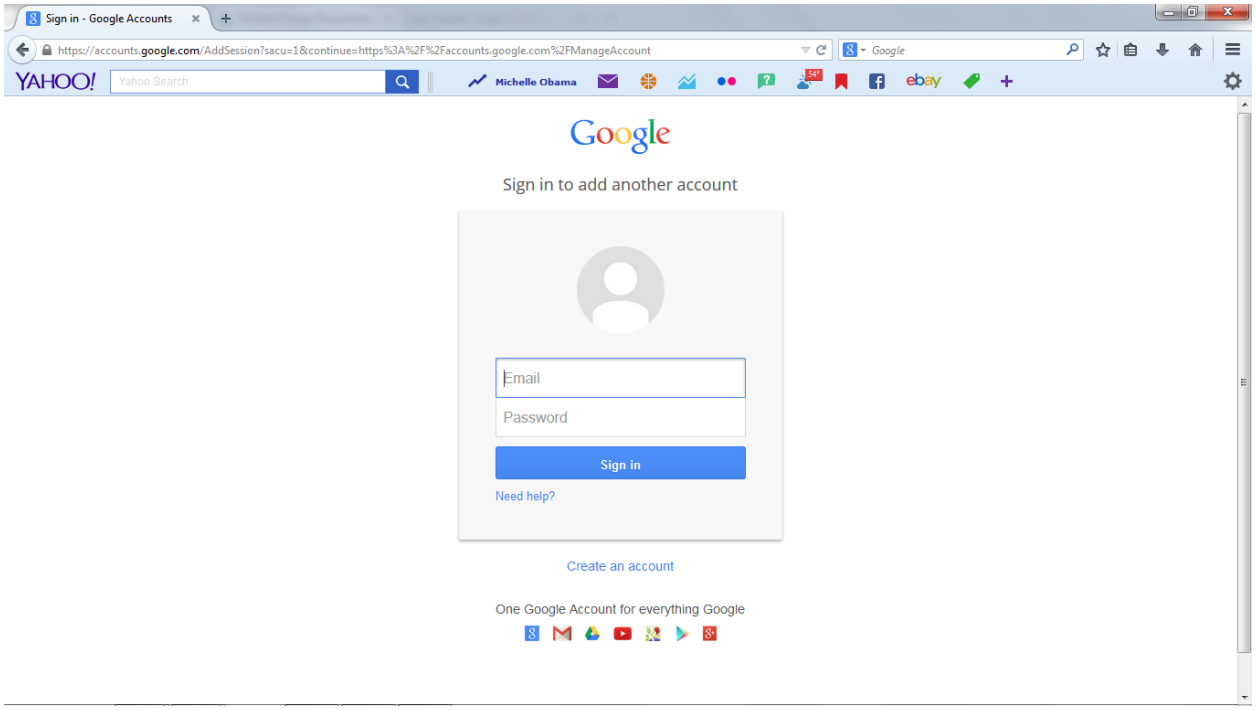
The Detailed Design Document covers our design choices for both front and back ends. For the front end, screen shots are provided which contain designs for each screen that the user will see. Various models and tables describe the processes of the back end.

GUI (Graphical User Interface) Design

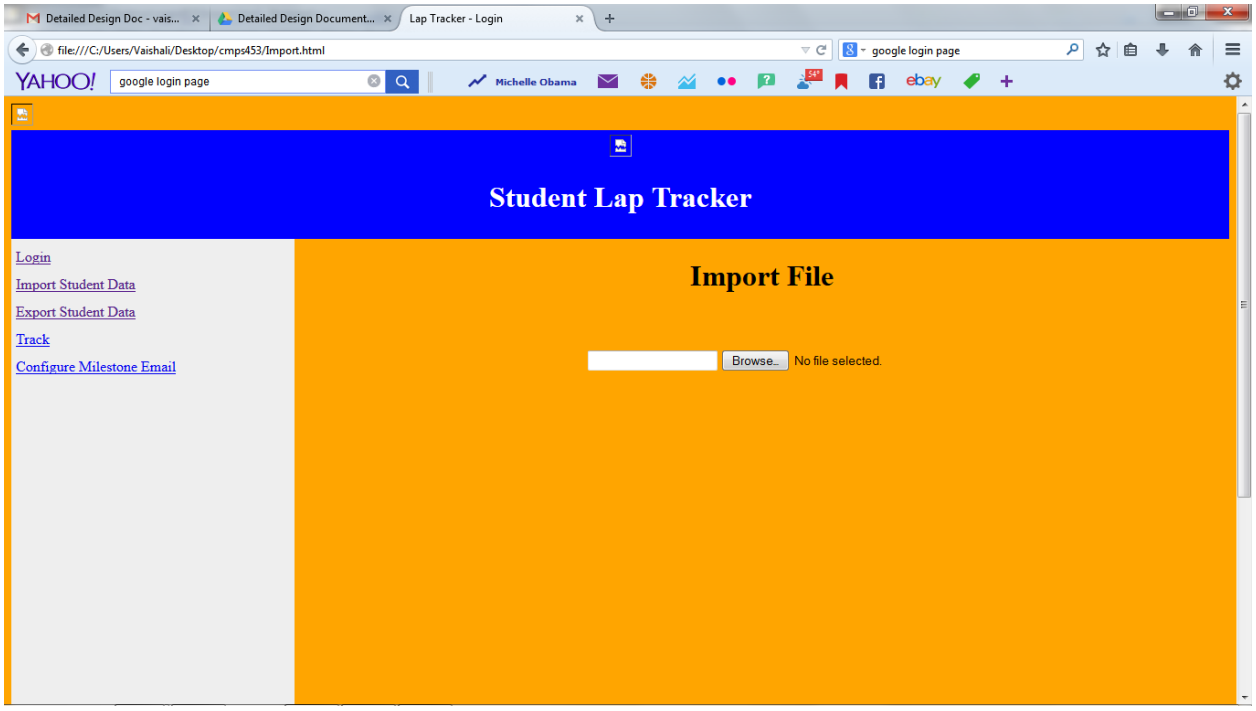
- **Home Page**: Contains the school logo and information as well as a button linking to the Google Account Sign-In.



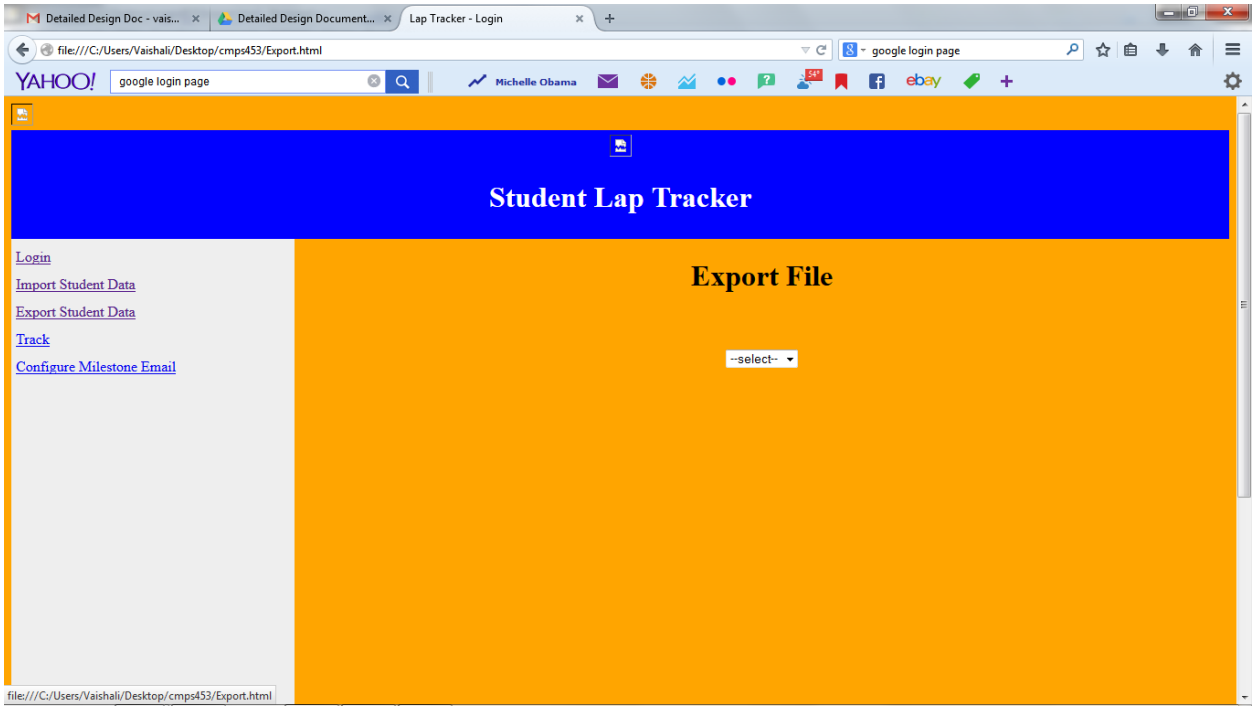
Login Page: Google provides our login page. Only an admin will be able to access all of the pages.



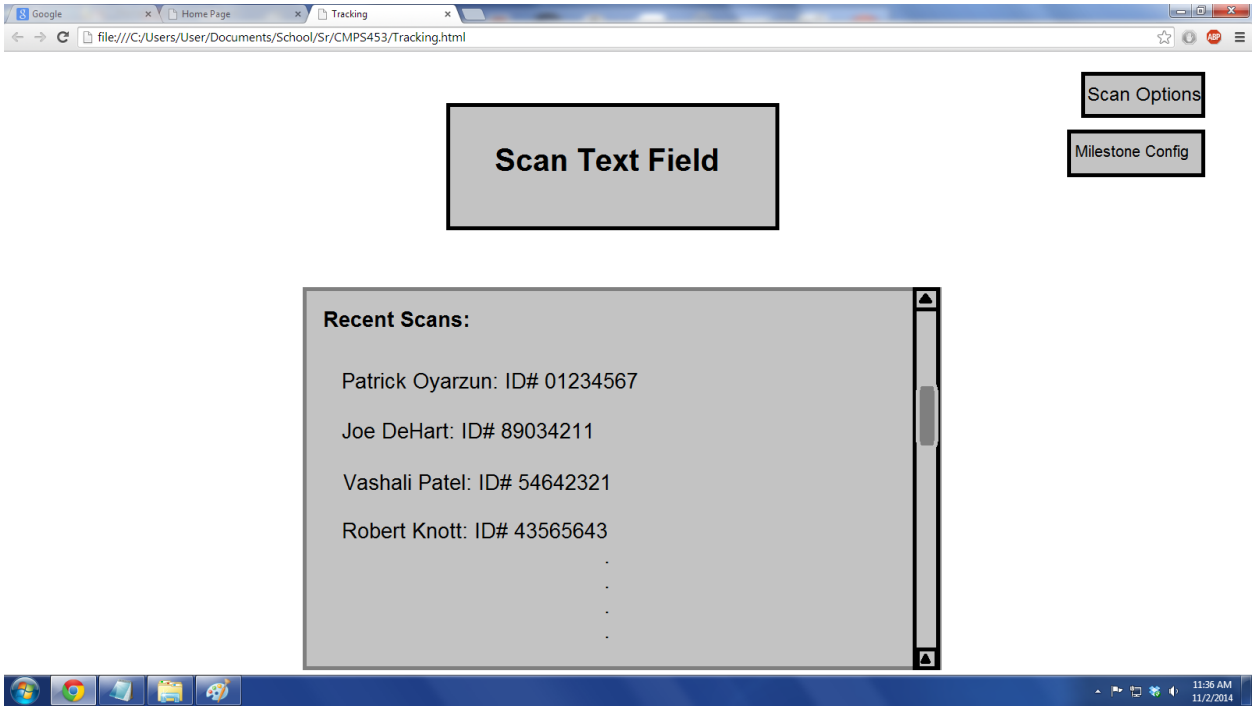
Import Page: A user can imports an Excel Spreadsheet file with .xlsx, which contains the students data through the browse button and the data of the selected file will be displayed on the page.



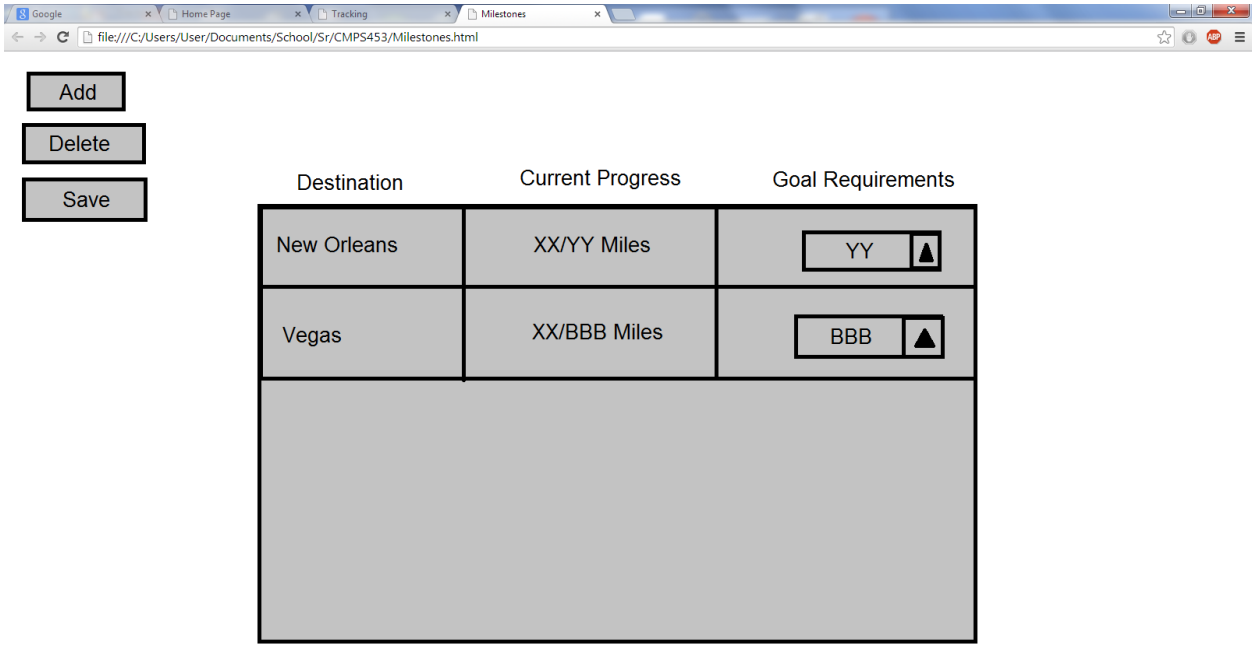
Export File: An export page will either display the data of all the students or the data of a particular student according to the selection of a drop down menu.



Tracking and Scan Page: The user navigates to this page to scan barcodes that will send upload requests to the App Engine Data Storage. There is also a Recent Scans window that will show a record of the most recent scans made. There are also two buttons used for modifications; Scan Options which will allow the user to define which course should be used to increment each ID scan, and Milestone Config, which allows the user to modify their Milestone goals.



Milestone Config: This page allows the user to add, delete, or modify any of their currently active milestones. It will send modification requests every time the Save button is pressed, and will automatically send one when the Add or Delete functions perform successfully. Should a Milestone be reached, it will automatically send a request to server that will e-mail the linked email address about the accomplishment.



// screen designs (coded or using drawing tool)

STATIC MODEL

- **CLASS DIAGRAMS**

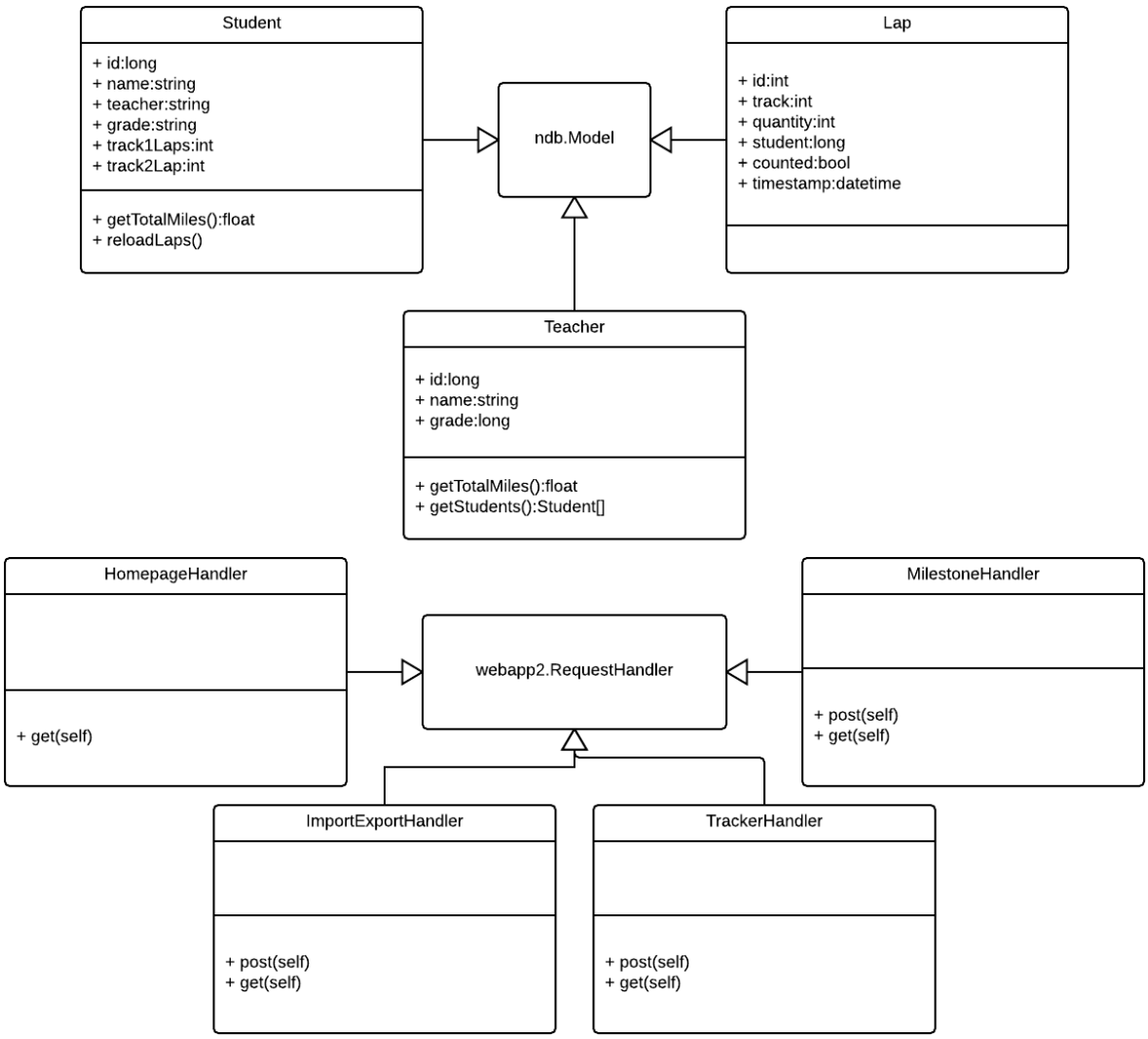


Figure 1. Class Diagram

Data persistence model:

The database schema is composed of three models. Due to the nature of the appengine datastore, there is no explicit schema enforced by the database. Instead each entity “Kind”, analogous to an RDBMS table, is defined by a normal python class definition which subclasses the ndb.Model class. All operations are then inherited by the subclass automatically. The three models are as follows:

Teacher

ID	Unique Identifier assigned by the appengine infrastructure

Name	Teacher's name taken from an uploaded excel spreadsheet
Grade	Teacher's grade taken from an uploaded excel spreadsheet

Table 1. Teacher Entity

Student

ID	Unique Identifier assigned by the school
Name	Student name taken from uploaded Excel spreadsheet
Teacher	The student's teacher, obtained from uploaded data
Grade	The student's grade, obtained from uploaded data
Track1Laps	The most recent sum of the student's laps on track 1, this value is maintained to speed up later calculations involving the total miles a student has walked.
Track2Laps	The most recent sum of the student's laps on track 2, this value is maintained to speed up later calculations involving the total miles a student has walked.

Table 2. Student Entity

Lap

ID	A unique identifier assigned by the appengine infrastructure
Track	Either 0 or 1, specifies which track was walked.
Quantity	The number of laps walked. Most instances will report 1 for this field, but in certain circumstances data is uploaded in aggregate and therefore there is no way to separate each lap into its own instance.
Student	The student who walked the lap's ID.

Counted	A boolean indicating whether or not this row has been counted in the student's total fields. This exists to speed up queries elsewhere in the system.
Timestamp	The time that the lap was completed. This is unused as of yet, but may be useful in the future.

Table 3. Lap Entity

DYNAMIC MODEL

- **IMPORT SEQUENCE**

The import flow involves three major steps. First, the data is uploaded to the server through an HTML form in the client implementation in the form of a Microsoft Excel spreadsheet. This data is then passed to a reader library called xlrd, which is part of the python-excel project [1]. The resulting workbook object is used to construct a set of database model instances which represent the uploaded data. Once these instances are constructed, they are inserted into the appengine datastore in a batch put operation which allows all the writes to occur in parallel on the distributed infrastructure. The user is then redirected to a summary page which gives an overview of the uploaded data so that the user can verify the import process was successful. In certain cases an import may fail due to corrupt data, service unavailability, etc. In any case where the error is detectable by the server code, an informative error message is shown instead of the summary page.

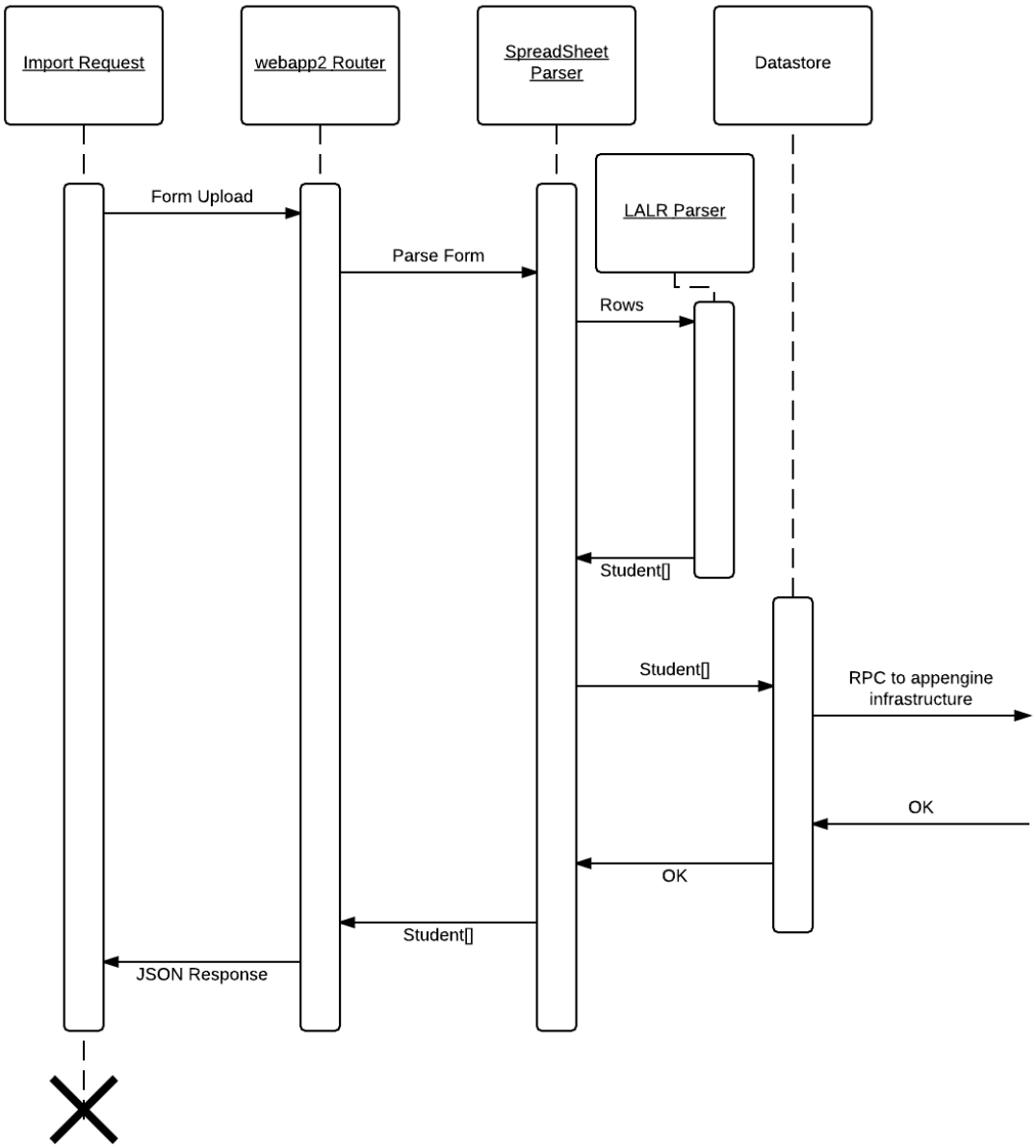


Figure 2. Import Sequence

- **EXPORT SEQUENCE**

The export sequence is simpler than the import flow since the data is in a predictable, structured format inside the appengine datastore. Therefore there are two main stages. First, a series of database queries are executed in order to obtain the relevant data depending on the subset of data the user is interested in exporting. This may be a system-wide backup, or a single student. That data is then used in conjunction with the xlwt library from the python-excel project [1] to construct the workbook object. This object can then be serialized to the http response.

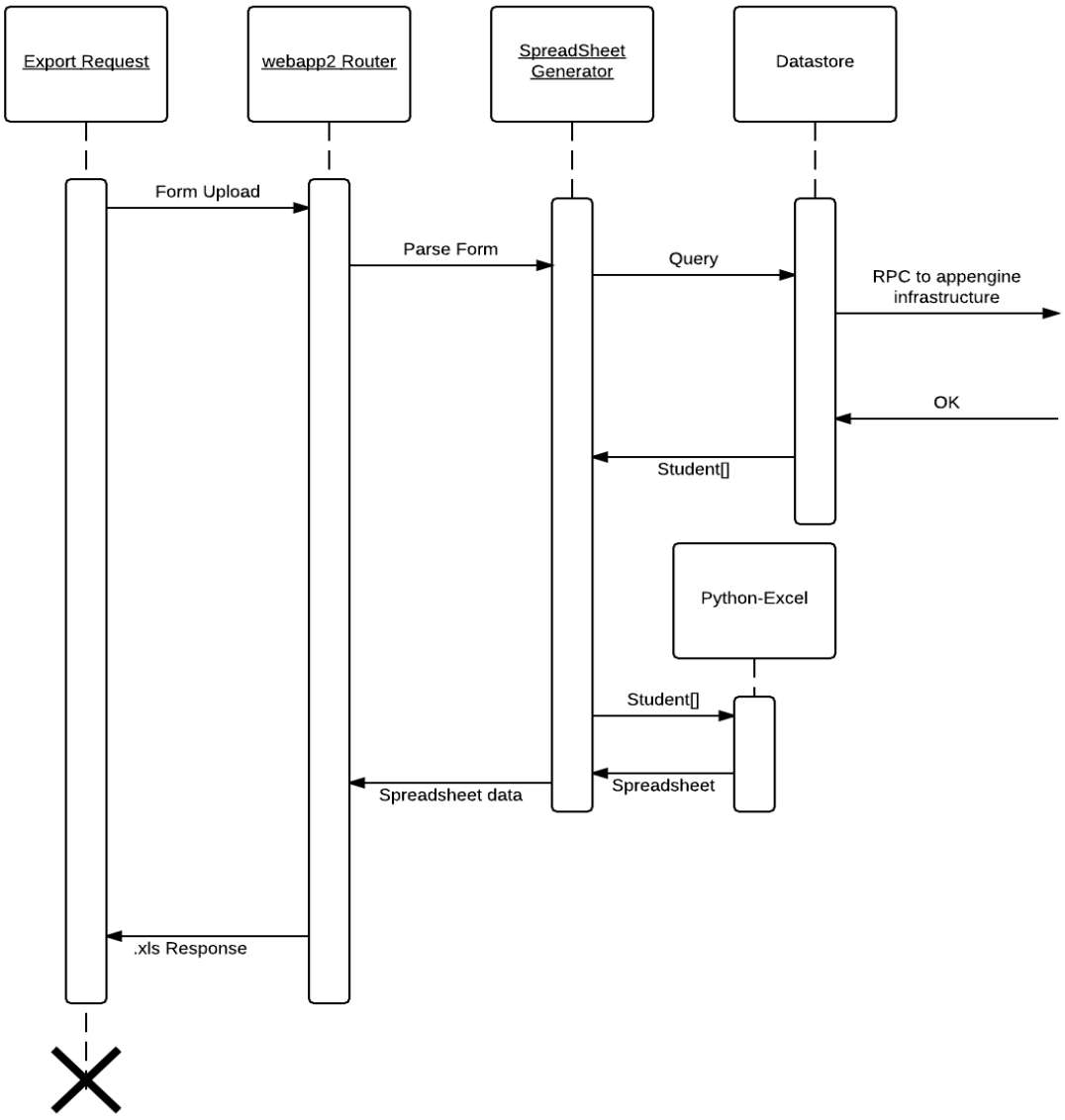


Figure 3. Export Sequence

RATIONALE FOR YOUR DETAILED DESIGN MODEL

Import / Export Rational

A primary goal of this project is to provide a minimum cost solution to the user. We are given a $0.00 budget to work with, so it is important that we cut any and all costs wherever feasible. The primary source of cost in many appengine applications is datastore usage. Billing is usage-based, and all applications receive a quota on each available resource under which there is no charge. For the appengine datastore, this quota is 50,000 read operations, 50,000 write operations, and 50,000 'small' operations [2]. Read and write are self-explanatory. A small operation is an operation that can be executed cheaply by google's infrastructure. This includes things like fetching an entity directly by its ID. In order to minimize our use of the datastore without losing any data, we have devised a caching strategy which makes effective use of the appengine distributed memcache as well as a few extra properties on the student and lap models to optimize our queries. Each time a lap is entered into the system, a single row is generated in the datastore. Its counted field is marked false to indicate that it has not yet been added to its student's lap totals. Each day as the milestone cron job is executing, all lap rows which have not yet been counted are collected and their quantities are added to the relevant students. This ensures there will be no contention when adding laps in bulk since each one is completely independent. It also ensures that we can quickly approximate any student's total mileage as well as the system mileage without having to perform an expensive sum operation on the lap table.

TRACEABILITY FROM REQUIREMENTS TO DETAILED DESIGN MODEL

The requirements documentation lists three primary use cases, Import, Export, and Track. Above are listed sequence diagrams for import / export. Also listed is a specification of the data model and rationale involved in implementing the lap tracking use case.

EVIDENCE THE DESIGN MODEL HAS BEEN PLACED UNDER CONFIGURATION MANAGEMENT

REFERENCES

[1] Withers, Chris. Working with Excel Files in Python [Online]. Available: http://www.python-excel.org/

[2] Quotas - Google App Engine -- Google Cloud Platform [Online]. Available: https://cloud.google.com/appengine/docs/quotas