

California State University, San Bernardino

CSUSB ScholarWorks

Theses Digitization Project

John M. Pfau Library

2013

Gradeboard: A cloud-based solution for a student grading system

Manoj Kulkarni

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd-project>



Part of the [Databases and Information Systems Commons](#), and the [Educational Technology Commons](#)

Recommended Citation

Kulkarni, Manoj, "Gradeboard: A cloud-based solution for a student grading system" (2013). *Theses Digitization Project*. 4183.

<https://scholarworks.lib.csusb.edu/etd-project/4183>

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

GRADEBOARD

A CLOUD-BASED SOLUTION FOR A STUDENT GRADING SYSTEM

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Manoj Kulkarni

March 2013

GRADEBOARD

A CLOUD-BASED SOLUTION FOR A STUDENT GRADING SYSTEM


A Project
Presented to the
Faculty of
California State University,
San Bernardino

by

Manoj Kulkarni

March 2013

Approved by:


David Turner, Chair, Computer Science and
Engineering

3/12/2013
Date


Ernesto Gomez


Richard J. Botting

© 2013 Manoj Kulkarni

ABSTRACT

Cloud-based systems have become increasingly popular because of their scalability and resource management techniques. It is now possible to build a huge online system in a short time without investing too much on computer hardware. Cloud systems provide a subscription-based model and efficient resource management where nodes are automatically shutdown when there is less or no traffic and automatically scaled up as traffic increases. Some of the very popular cloud platforms include Amazon Web Services, Microsoft Azure and Google App Engine. The student grading system implemented in this project (GradeBoard) uses Google App Engine for cloud storage and application hosting. This application allows students and instructors to interact using desktop or mobile platforms.

GradeBoard uses the JQuery Mobile library, which enables deployment to multiple mobile platforms, such as Android and IOS operating systems, and on desktop systems without having to build separate applications for each of these platforms. This reduces the cost of development and maintenance when supporting multiple platforms.

The GradeBoard application is developed using the Model View Controller architecture. It consists of server side and client side components that interact with each other using HTTP and HTTPS protocols. The data is stored in Google App Engine Datastore. The server side components are developed using Java Servlets, which connect to Google App Engine Datastore for retrieving and updating the data. The client side components are developed using JQuery Mobile library. They mainly consist of User Interface(UI) components and a Javascript library for interacting with the server side components using Asynchronous JavaScript and XML (Ajax).

Overall, cloud platforms such as Google App Engine provide advantages by reduce the cost of hardware, facilities, and human resources. As a result, there will likely be a lot of applications migrated to, or built on top of, cloud platforms in coming years. Additionally, with the increasing number of mobile and tablet users, applications such as GradeBoard that run on both mobile and desktop platforms are becoming essential.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. David Turner for his invaluable guidance, advice, support, help, and patience during this project's long gestation. I would also like to thank my committee advisors Dr. Richard J. Botting and Dr. Ernesto Gomez for their valuable reviews and advice. I would like to thank all the people with whom I have worked while pursuing my master's degree at California State University, San Bernardino (CSUSB). I wish I could list all their names but the list would be too long and I would still probably leave some people out. Studying in the School of Computer Science and Engineering at CSUSB has been a tremendous learning experience, both personally and professionally.

I would also like to thank my wife for her undying support for my studies over the years. Finally, thanks to my parents who encouraged me all along.

TABLE OF CONTENTS

<i>Abstract</i>	iii
<i>Acknowledgements</i>	v
<i>List of Tables</i>	ix
<i>List of Figures</i>	x
1. Introduction	1
1.1 Background	1
1.2 Google App Engine	2
1.3 JQuery Mobile	3
1.4 Java Servlet Application Program Interface	3
1.5 Purpose	3
1.6 Project Scope	4
1.7 Related Work	5
1.8 Project Limitations	5
1.9 Definitions, Acronyms, and Abbreviations	5
2. Architecture and Design	9
2.1 Design Overview	9
2.2 Model View Controller Architecture	9
2.3 Detailed Design	12

2.4	Application Components	13
2.5	Controller Classes	13
2.6	Mapping of Model Classes to Google App Store Objects	14
2.7	System Interfaces	14
2.8	Product Functions	15
3.	<i>Google Data Store Design</i>	19
3.1	Entity and Kind	19
3.2	Indexes	20
3.3	Data Modeling	21
4.	<i>Project Implementation</i>	26
4.1	Welcome Screen	27
4.2	Login Screen	28
4.3	Quarter Screen	29
4.4	Courses Screen	30
4.5	Add Course Screen	31
4.6	Course Details Screen	32
4.7	Edit Course Screen	33
4.8	Edit Course Name Screen	34
4.9	Instructors Screen	35
4.10	Add Insructor	36
4.11	Delete Insructor	37
4.12	Students	38
4.13	Add Student	39
4.14	Add Multiple Students	40
4.15	Edit Student	41
4.16	Grades	42

4.17 Edit Grades	43
4.18 Edit Student Grade	44
4.19 Gradable Components	45
4.20 Add Gradable Component	46
4.21 Edit Gradable Component	47
5. <i>Conclusion and Future Direction</i>	48
5.1 Conclusion	48
5.2 Future Direction	48
<i>APPENDIX A: SERVER SOURCE CODE</i>	50
<i>APPENDIX B: CLIENT SOURCE CODE</i>	75
<i>References</i>	86

LIST OF TABLES

3.1	Entity Types	21
3.2	Entity Path	22
3.3	Quarter Entity	22
3.4	Course Entity	23
3.5	Component Entity	23
3.6	Grade Entity	24
3.7	Student Entity	24
3.8	Instructor Entity	25
3.9	Admin Entity	25

LIST OF FIGURES

2.1	Model View Controller Architecture In GradeBoard Application . . .	12
2.2	Deployment Diagram	15
2.3	Admin Use Case Diagram	16
2.4	Instructor Use Case Diagram	17
2.5	Student Use Case Diagram	18
4.1	GradeBoard Welcome Screen	27
4.2	Login Screen	28
4.3	Quarter Screen	29
4.4	Courses Screen	30
4.5	Add Course Screen	31
4.6	Course Details Screen	32
4.7	Edit Course Screen	33
4.8	Edit Course Name Screen	34
4.9	Instructors Screen	35
4.10	Add Instructor Screen	36
4.11	Delete Instructor Screen	37
4.12	Students Screen	38
4.13	Add Student Screen	39
4.14	Add Multiple Students Screen	40
4.15	Edit Student Screen	41
4.16	Grades Screen	42

4.17 Edit Grades Screen	43
4.18 Edit Student Grade Screen	44
4.19 Gradable Component Screen	45
4.20 Add Gradable Component Screen	46
4.21 Edit Gradable Component Screen	47

1. INTRODUCTION

1.1 Background

Hosting a web site and maintaining servers requires a lot of resources such as hardware, air conditioned facilities, networking, electricity, human resources for system maintenance, etc. When a website is initially hosted, there are very few users using the system. But as the number of users increase, hardware needs to be upgraded along with building new facilities to accommodate the systems and human resources to maintain these systems. Sometimes, this new requirement of upgrading resources has to be met in a very short amount of time, which may be practically impossible. Also, if the demand drops suddenly after upgrading, then the entire investment on upgraded resources becomes wasted.

Cloud computing has evolved to overcome some of these problems. Cloud computing involves providing the software and hardware services based on the subscription-based model. These services are provided over the Internet.

Hosting of a website on a cloud platform requires subscribing into these services based on the number of users and resources. These subscriptions can be upgraded within a short of amount time as opposed to days waiting for the shipment and installation of new hardware. Also, subscriptions can be downgraded without having to waste additional resources. The sudden spike in demand or drop in demand can be immediately addressed by using the auto scaling feature of a cloud platform. The auto scaling features of cloud platforms allow the spawning of new system resources as demand increases within minutes and at the same time shutting down resources

as demand decreases. This saves lot of time, hardware costs and electricity.

Some of the popular cloud system platforms include Amazon Web Services, Google App Engine, Microsoft Azure, and Heroku, to name a few.

Nowadays web applications need to be supported on multiple platforms such as desktops, laptops, mobile devices and tablets. Building a native application for each of these platforms is very time consuming and expensive. On the other hand, applications that are built using mobile development frameworks, such as PhoneGap and JQuery Mobile, work on multiple platforms without having to develop applications using native APIs. This provides developers with a cost effective way to create applications that work on multiple devices.

The GradeBoard project is built by using Google App Engine for the Java platform and JQuery Mobile.

1.2 Google App Engine

Google App Engine is a cloud-based system that provides an application hosting environment that includes auto scaling, load balancing, security, a backend data store, and various other useful services. Google App Engine provides an API for languages such as Python, Java, and Go. Typically developers build and test their applications locally, before deploying into the Google App Engine hosting environment. Google App Engine provides an Eclipse IDE plug-in for Java developers to debug and test their applications. Google App Engine also provides an administrative console for monitoring performance, traffic and errors, etc [10].

The GradeBoard application uses the Google App Engine Datastore to take advantage of object data stores, query engines and atomic transactions [23].

1.3 JQuery Mobile

JQuery mobile is an HTML5 mark up language built on top of JQuery and JQuery UI foundations. The primary purpose of JQuery Mobile is to support applications on various platforms such as desktops, mobile phones, and tablets. JQuery Mobile provides an API for developers to create rich HTML web pages using the JQuery markup, eventing model, AJAX API for asynchronous communication with Web servers and other useful features such as page transitions, accessibility, touch and slide support, etc [18].

The GradeBoard application uses JQuery Mobile API as the front end to interact with the Google App Engine.

1.4 Java Servlet Application Program Interface

Java Servlet Application Program Interface(API) is a Java extension to process requests and responses from a Java Servlet Engine running inside a Web server. The communication between a client and the Web server is done using the HTTP or HTTPS protocol. Google App Engine internally uses Jetty Web server for processing HTTP and HTTPS requests. The GradeBoard is developed using Java Servlet API for handling requests and responses on the server side.

1.5 Purpose

The purpose of the GradeBoard application is to develop a Mobile application as a front end to interact with data stored in a cloud-based system. It leverages cloud computing features such as auto-scaling, load-balancing and NoSQL data stores. The GradeBoard application helps instructors keep track of student activities and gives students an opportunity to know how they are performing by comparing their grades with other students through a game-like leader board interface, thus creating a com-

petitive environment in the class. The goal is to encourage students to spend more time in studies and score better grades.

1.6 Project Scope

The primary users of this system are instructors and students. Instructors use the system to maintain or keep track of student activities and students use the system to view the grades of students in the course and compare their grades with those of their classmates. There is also an admin user for overall management of the student grading system.

The Gradeboard application provides the following functionalities:

- Admin manages courses and instructor access to them.
- Instructors login to the system to populate courses with registered students.
- Courses can have more than one instructor.
- Instructors create gradable components for their courses.
- Instructors record student grades on gradable components.
- Instructors email links of public grade sheets to students after grade changes.
- Instructors mask the student ids with randomly generated strings of length 2-3 characters.
- Grade sheets are public; students do not authenticate to view their grades.
- Instructors and admin authenticate using the Google authentication service.
- Students can view the grade sheet on mobile devices or desktop computers.
- The grade sheet is designed to function well on mobile devices.
- Instructors can set letter grade point ranges.

1.7 Related Work

Currently there are not many solutions available that provides a student grading system along with a Mobile interface for accessing the data in the cloud. The GradeBoard application is robust because of the use of Java Servlet technology. Java Servlet technology is a widely adopted technology. Most of the student grading systems that are currently available in the market are not designed to scale easily to match demand, and lack UI interfaces that are developed for mobile devices and tablets. The GradeBoard application provides both cloud and mobile based solutions along with a leader board interface for students to evaluate their performance.

1.8 Project Limitations

This project provides a UI design for mobile devices and tablets. A responsive design spanning from desktop to mobile devices with screen resizing and scaling can not be fully provided because of the limitations of the JQuery Mobile API. Monitoring an application hosted on a Google App Engine is not available for free for developers.

1.9 Definitions, Acronyms, and Abbreviations

The definitions, acronyms, and abbreviations used in the document are described in this section.

- GradeBoard: Name of this project
- API: Application Programming Interface is a set of routines that an application uses to request and carry out low-level services performed by a computer's operating system; also, a set of calling conventions in programming that defines how a service is invoked through the application [9].

- Cloud computing: Cloud computing is the use of computing resources (hardware and software) that are delivered as a service over a network (typically the Internet) [6].
- Google App Engine: Cloud computing platform provided by Google [10].
- JQuery: A javascript library provided by JQuery for building web based applications [19].
- UI: User Interface
- JQuery Mobile : A Mobile API built using JQuery and JQuery UI foundation [18].
- Java: An Object Oriented Language developed by Sun Microsystems in 1995 [25].
- Java Servlet: Java Servlet technology provides Web developers with a simple, consistent mechanism for extending the functionality of a Web server and for accessing existing business systems [24].
- CSUSB: California State University, San Bernardino.
- HTML: HyperText Markup Language is the authoring language used to create documents on the World Wide Web [29].
- HTTPS: Hyper Text Transfer Protocol Secure is a secure network protocol used to encrypt data transferred between server and client [13].
- J2EE: Java 2 Platform Enterprise Edition is a platform-independent, Java-centric environment from Sun Microsystems, Inc., for developing, building, and deploying Web-based enterprise applications online. The J2EE platform consists of a set of services, APIs, and protocols that provide the functionality for developing multitiered, Web-based applications [30].

- JDBC: Java Database Connectivity is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases [16].
- MVC: Model-View-Controller is an architectural pattern used in software engineering to isolate business logic from user interface considerations [21].
- UML: The Unified Modeling Language is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems [28].
- Microsoft Azure: Cloud Computing platform provided by Microsoft [32].
- Amazon Web Services: Cloud Computing platform provided by Amazon [3].
- Heroku: Cloud Application platform provided by Heroku [11].
- Android : Mobile Operating System provided by Google [4].
- IOS : Mobile Operating System provided by Apple [15].
- PhoneGap: Open Source Framework for creating Mobile Apps [26].
- Google Cloud SQL: Relational Database Model for Cloud Platform provided by Google [23].
- Google Cloud Storage: Cloud Storage Service provided by Google [23].
- Eclipse IDE: Open Source Development Environment provided Eclipse Foundation [7].
- NoSQL: Uses key-value pairs for storing data unlike traditional Relational Database Management [22].

- Jetty (web server): An open source HTTP server and a Java Servlet container [17].
- JSON : Javascript Object Notation built using key and value pairs [20].
- Ajax: Asynchronous JavaScript and XML/JSON format for communicating from client to the server [2].
- Gradable componet: Student work receives grade, quiz exam assignment.
- Grade Sheet: A tabular format with columns representing gradable components and rows resprsenting students.Cells contain assigned grades.
- OOP : Object Oriented Programming concept with objects representing real world entities. Methods expose state of the object [31].
- PolyModel : Object Oriented Programming concept built into Google App Engine Datastore [27].

2. ARCHITECTURE AND DESIGN

2.1 *Design Overview*

GradeBoard uses a client-server architecture model. On the server side, it uses the Google App Engine Datastore for storing data and Java Servlets running inside Jetty Web Server for serving requests and responses sent over HTTP(S). On the client side, it uses JQuery Mobile API for building UI components and for communicating with the Web server. The requests and responses on the client side are handled through Asynchronous Javascript and XML (Ajax).

2.2 *Model View Controller Architecture*

Model View Controller (MVC) architecture is a software design pattern for separating different components of a software application [21]. There are three main categories in the MVC architecture:

- Model represents data in the application. All the business rules are handled in the model.
- View represents UI components in the application. The UI components are responsible for presenting the model and for collecting user inputs.
- Controller is responsible for updating the data in the model and notifying the view about changes in the model.

There are several advantages of using an MVC architecture:

1. Easy to maintain large applications.
2. Easy to identify bugs in the system.
3. Developers can work on different components based on their expertise.
4. Reduces clutter in the application logic as the application grows in size.

In this project, an MVC architecture is used in both server side and in client side components.

On the server side, all the business rules are implemented using plain Java classes which represent the model. These Java classes are responsible for communicating and updating the Google App Engine Datastore. The Java Servlets act as a controller for dispatching requests and updating the model. There are no view components on the server side.

On the client side, all business rules are implemented using Javascript objects which represent the model. All the Ajax requests are handled through a controller class. The rendering of HTML markup is handled through a view class.

In this design, there is a one-to-one mapping between the model represented on the client side with the model represented on the server side. Also, there is a one-to-one mapping between the model represented on the server side with the data stored in the Google App Engine Datastore.

A typical workflow of processing user request is as follows:

- When a page is requested, the controller is initialized and sends an Ajax request to the server.
- The request arrives on the server side; the Servlet controller parses the input parameters and sends the request to the model.
- The model processes input parameters, connects to the Google App Engine Datastore and queries the data.

- The model sends the response back to the Servlet controller.
- The Servlet controller dispatches the response back to the client.
- The controller on the client side receives the response and updates the model with new data.
- The controller calls on the view to render the data.
- The view gets the data from the model and renders the data.
- The user views, modifies or inputs new data and submits the request. The controller sends the request to the server and the cycle continues.

The MVC architecture used in GradeBoard for processing user requests is shown in Figure 2.1.

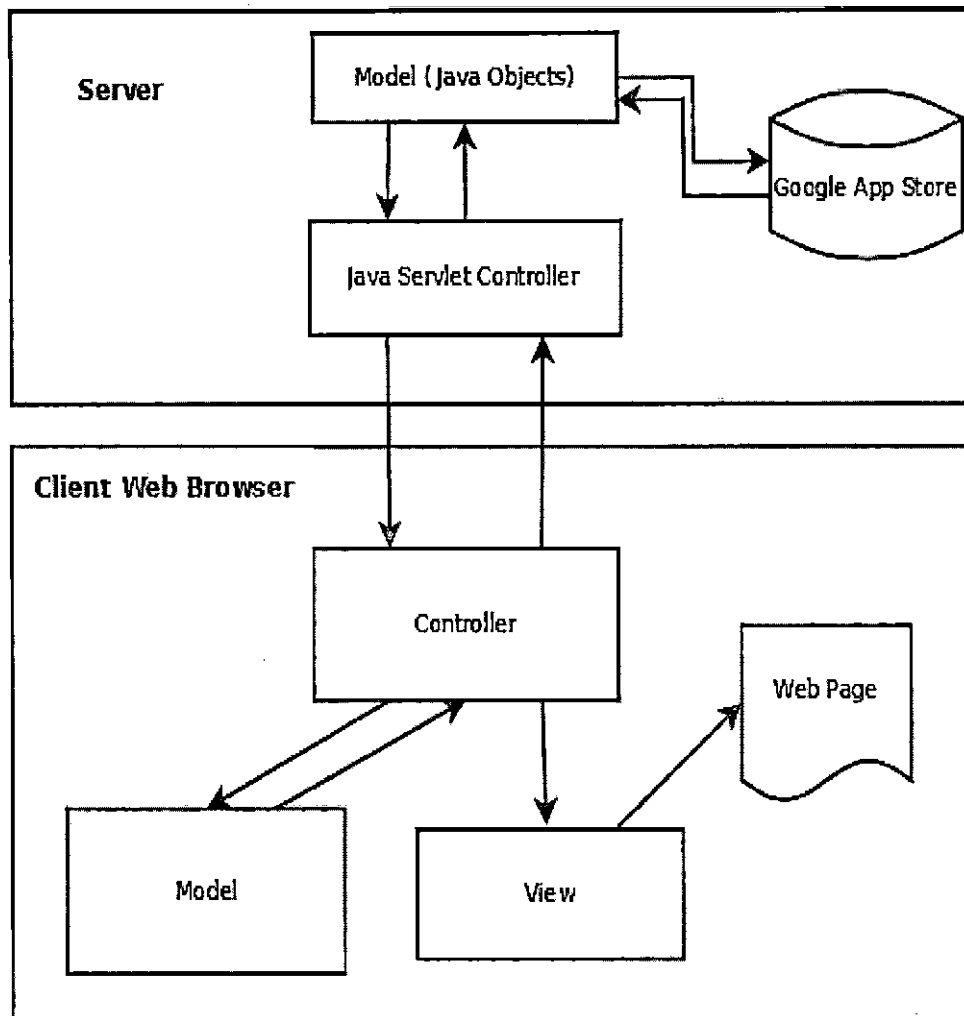


Fig. 2.1: Model View Controller Architecture In GradeBoard Application

2.3 Detailed Design

Besides using an MVC model, the GradeBoard application design completely separates server and client side components. There are no JSP pages for rendering HTML markup from the server side. This gives the flexibility of rendering UI components using the Javascript libraries such as JQuery Mobile. The data communicated between client and server is constructed using JavaScript Object Notation (JSON). JSON

provides light weight data communication medium between server and client usually through Ajax [20].

2.4 *Application Components*

The server side components include the controller and model classes. Controller classes extend from the Java HTTP Servlet class. There are two main controller classes, StudentController and InstructorController, for handling HTTP and HTTPS requests. These controller classes can be thought of as dispatchers for parsing inputs from the HTTP Servlet request object and appropriately calling model objects.

2.5 *Controller Classes*

The InstructorController class processes the following requests:

- Adding, Deleting and Modifying instructors.
- Creating, Modifying and Deleting a course.
- Creating, Modifying, Deleting Gradable components for the course.
- Modifying Grades.
- Generating Grade Sheet.

The StudentController class processes the following requests:

- Delete, Modify and Update students.
- Allow admin to enter multiple students for a course.
- Email student grades

2.6 Mapping of Model Classes to Google App Store Objects

The GradeBoard application uses a one-to-one mapping between Google App Engine Datastore object representations to model classes. Besides these mapping, objects in Google App Store can be represented in a hierarchical model by having an ancestor class. This hierarchical architecture can be easily represented in model classes by using inheritance. The Model classes used in this project include:

- Course class for Creating, Modifying and Deleting a course.
- Component class with Course as an ancestor.
- Grade class with Component as ancestor for creating Gradable components.
- Instructor class for storing instructor information.
- Auth class for storing authorized Instructor for a course with instructor as an ancestor.
- Student class for adding, modifying and deleting students with Course as ancestor.

Internally, the model classes communicate with Google App Engine Datastore objects by constructing queries from request objects.

2.7 System Interfaces

GradeBoard application users interact with the system using a Web browser on a desktop computer or a mobile device. The requests are sent over the network using HTTP or HTTPS protocols. On the server side, GradeBoard application is deployed on a Jetty Web server. When a Web server receives the request, it calls appropriate application class to process the request. If the request is for data from Google Datastore, then a connection is established with Google Datastore to retrieve the data.

After retrieving the data, a response object is created and the data is transferred back to the Web browser through HTTP or HTTPS. The deployment diagram for the GradeBoard application is shown in Figure 2.2.

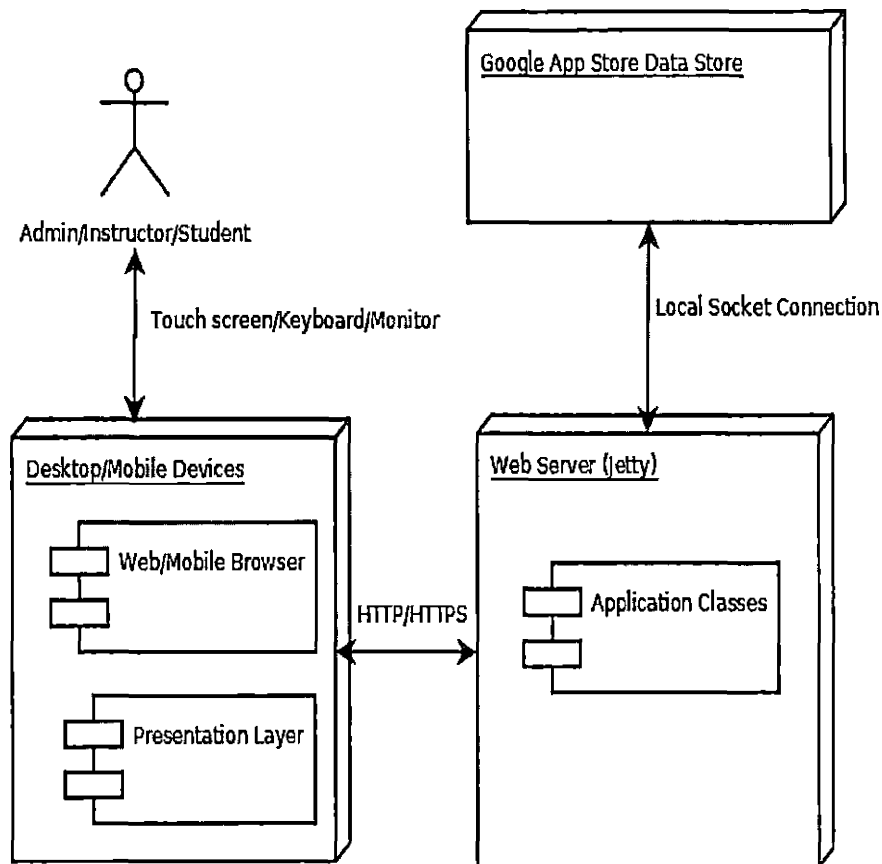


Fig. 2.2: Deployment Diagram

2.8 Product Functions

In the GradeBoard application there are three main users, namely instructor, student and admin.

Admin characteristics:

- Must be logged-in through Google Account as an admin user.
- Authorizes instructor to create courses.
- Admin adds students to specific courses.

The use case diagram for admin user is shown in Figure 2.3.

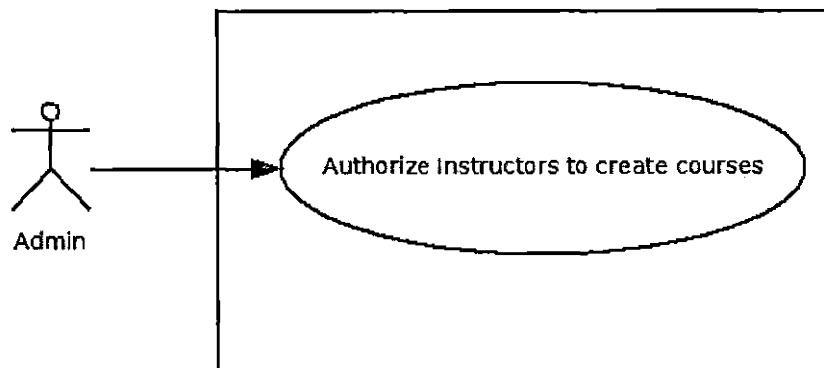


Fig. 2.3: Admin Use Case Diagram

Instructor characteristics:

- Must be logged-in through Google Account and authorized to modify or create courses.
- Creates courses.
- Authorizes another instructor to modify the courses.
- Creates Gradable components.
- Assign Scores to Gradable Component for specific students.
- Emails grade sheet URL to students.

The use case diagram for instructor is shown in Figure 2.4.

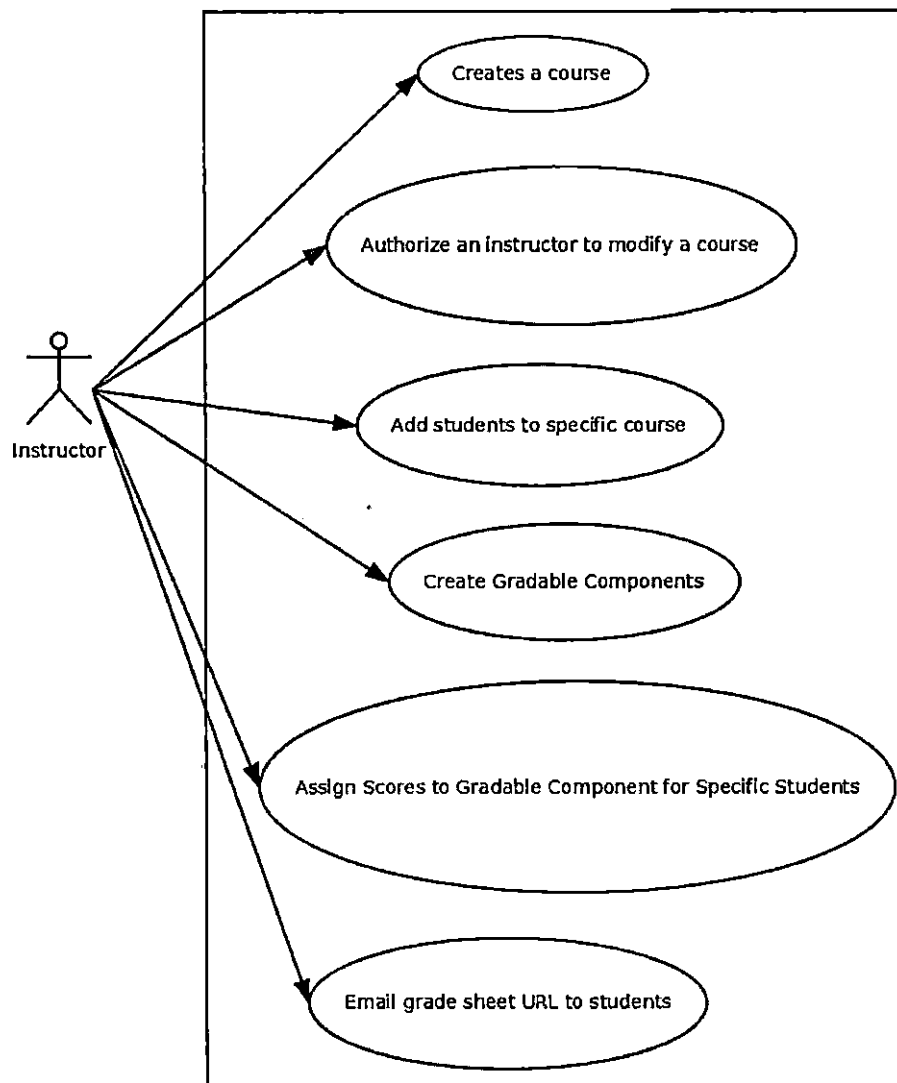


Fig. 2.4: Instructor Use Case Diagram

Student characteristics:

- Receives grade sheet URL through email.
- Views Grades for specific course.

The use case diagram for students is shown in Figure 2.5.

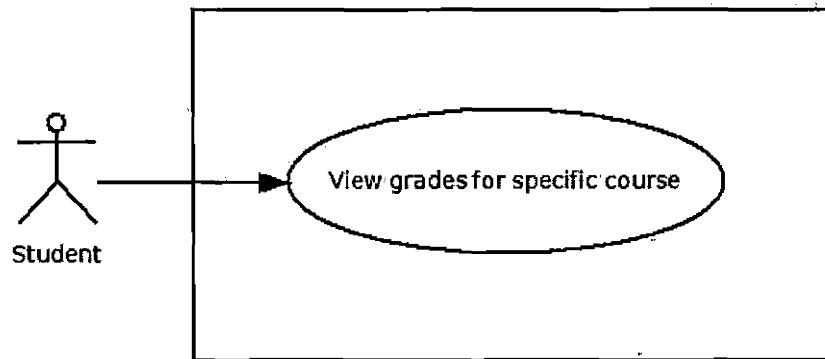


Fig. 2.5: Student Use Case Diagram

3. GOOGLE DATA STORE DESIGN

The Google App Engine Datastore is based on the Big Table architecture. It is essentially a hashmap with keys mapped to multi dimensional arrays. These arrays represent columns or collections of columns. The number of columns in a datastore is not fixed. For every web application deployed in Google App Engine, it generates a unique application id (ApplicationID). The application id (ApplicationID) is associated with a path, which defines the hierarchy of entity keys [12].

3.1 *Entity and Kind*

Every entity must be of a particular kind. For example: To create an entity of kind Department, an object of type Entity is created by passing Department as kind:

```
Entity department = new Entity("Department");
```

An entity can be associated with a parent entity by providing the entity type in the constructor.

```
Entity location = new Entity("Department", department.getKey());
```

Each entity may or may not belong to a group. But Google recommends every entity should belong to a group so that in a distributed environment, a group of entities can be stored in a particular cluster. Deleting a parent, does not guarantee deletion of the child. This should be done programmatically by traversing each child [8].

Each entity can have multiple properties. For example:

```
location.setProperty("USA", "");
```



```
location.setProperty("France", "");
```

3.2 Indexes

Indexes are auto generated in the Google App Engine Datastore. Each application contains an index table with rows corresponding to entities and columns corresponding to property values. Index entries are added based on the combination of the property name and property values. So care must be taken to avoid index explosion. For example:

Indexes

```
kind =Model;
  properties:
    name = type1
    name = type2
```

If there are three values of type1 and four values of type2, a combination of these will result in 12 index table entries. Google App Engine datastore allows 200 entries in the index table for every put operation. To avoid indexing errors, property names can be grouped into multiple entities.

Indexes:

```
kind =Model1:
  properties:
    name = type1
```

Indexes:

```
kind =Model2:
  properties:
    name = type2
```

Custom indexes can be added by updating indexing.yaml file by specifying which properties need to be indexed [14].

3.3 Data Modeling

GradeBoard uses the Google App Engine Datastore API for defining models. The attributes of an entity can be updated dynamically since there is no limitation on the number of columns. Table 3.1 defines entities identified in the GradeBoard application.

Tab. 3.1: Entity Types

Entity Kind	Keytype
Quarter	int
Course	int
Component	int
Grade	int
Instructor	int
Student	int
Auth	int

Each entity is defined in the datastore with an ApplicationID (unique across entire store) and a Path. The Path defines hierarchy beginning with the root entity until the current node is found. The relationship is identified by ancestor path. Path defines the ancestor relationships among entities. Table 3.2 defines paths identified in the GradeBoard application.

Tab. 3.2: Entity Path

Entity	Pathvalue
Quarter	""
Course	Course:Quarter
Component	Course:Component
Grade	Course:Component:Grade
Instructor	""
Student	Course:Student
Auth	Auth:Instructor

Table 3.3 defines Quarter Entity in the GradeBoard application.

Tab. 3.3: Quarter Entity

Attribute	Type
Id	int
Name	str
Startdate	date
Enddate	date

The course entity contains details about the course. Table 3.4 defines the course entities in the GradeBoard application.

Tab. 3.4: Course Entity

Attribute	Type
Id	int
Coursename	str
Description	str
Number	int
Units	int
Totalseats	int
Seatstaken	int
Classtime	date
Location	str

Component is a PolyModel Entity. A PolyModel entity can be thought of as an abstract class which can be inherited by multiple entities. The query on a parent results in the data stored in the inherited child entity. This concept is similar to polymorphism in object oriented programming. Table 3.5 defines Component Entity in the GradeBoard application.

Tab. 3.5: Component Entity

Attribute	Type
Id	int
Name	str
Deadline	date
Points	int

The grade entity inherits from component entity. Table 3.6 defines the grade entity in the GradeBoard application.

Tab. 3.6: Grade Entity

Attribute	Type
Id	int
Score	int
StudentId	ReferenceProperty

Table 3.7 defines Student entity in the GradeBoard application. Student entity contains a property named SecretId for masking the Id when a grade sheet for other students is constructed. A student who is viewing the grade sheet will see the SecretId of the other students not the actual name.

Tab. 3.7: Student Entity

Attribute	Type
Id	int
Firstname	str
Lastname	str
Email	str
SecretId	int

Table 3.8 defines the Instructor entity in the GradeBoard application. Instructor information is populated from the Google Account that the instructor uses to log in. Administrator authorizes and adds the instructor account for GradeBoard application.

Tab. 3.8: Instructor Entity

Attribute	Type
Id	int
GoogleUser	User

Table 3.9 defines the Admin entity in the GradeBoard application. The Admin entity is pre-populated or manually updated by the administrator.

Tab. 3.9: Admin Entity

Attribute	Type
Id	int
GoogleUser	User

4. PROJECT IMPLEMENTATION

The GradeBoard application is designed to work on mobile devices and desktop computers. The UI of the application is developed using JQuery Mobile API. When a page requires the data to be loaded from server or modified or deleted, a request is sent to the Web server over HTTP or HTTPS. The requests are sent to the Web server using Ajax. For handling Ajax requests and responses, this application uses JQuery Ajax API. All UI components are dynamically created or initialized in response to the data received from the Web server.

4.1 Welcome Screen

When the GradeBoard application is loaded, a welcome screen with login button is presented to the user as shown in the Figure 4.1. The welcome screen shows application logo and login button for users to login to the system.

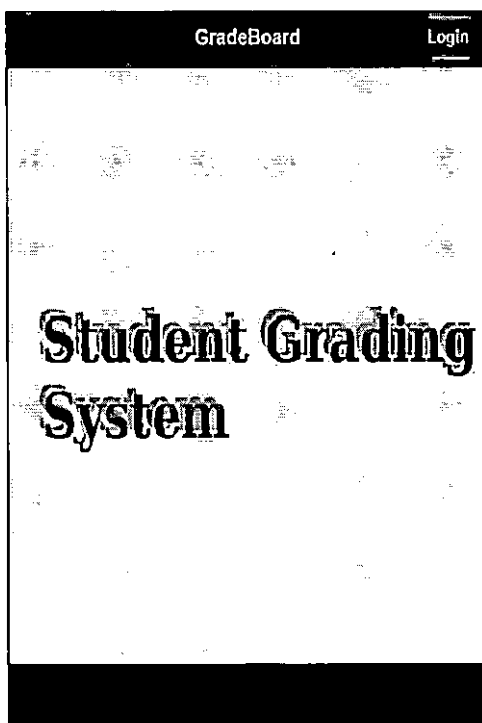
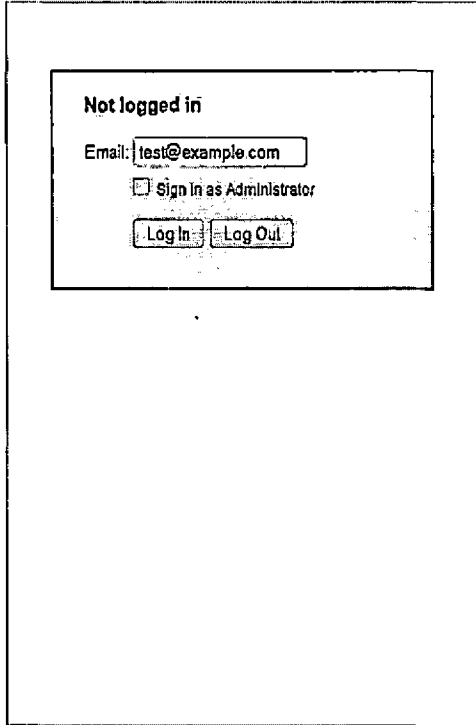


Fig. 4.1: GradeBoard Welcome Screen

4.2 Login Screen

GradBoard uses Google accounts for authentication and authorization. When the user clicks on the login button, the screen is automatically redirected to the Google login screen as shown in Figure 4.2. There are two types of access users can get, namely admin and user. For admin access, the user is first authenticated by Google and then the user entry is checked in the admin table. For user access, the user account is authenticated by Google. After logging in, the quarter screen is shown.



The image shows a login interface within a rectangular frame. At the top, it says "Not logged in". Below that is an "Email:" label followed by a text input field containing "test@example.com". Underneath the input field is a checkbox labeled "Sign In as Administrator". At the bottom of the form area are two buttons: "Log In" and "Log Out".

Fig. 4.2: Login Screen

4.3 Quarter Screen

The quarter screen displays quarter and year information as shown in Figure 4.3. The quarter screen is shown after user logs into the system. After selecting quarter and year, the instructor clicks on the show courses button to view the list of courses.

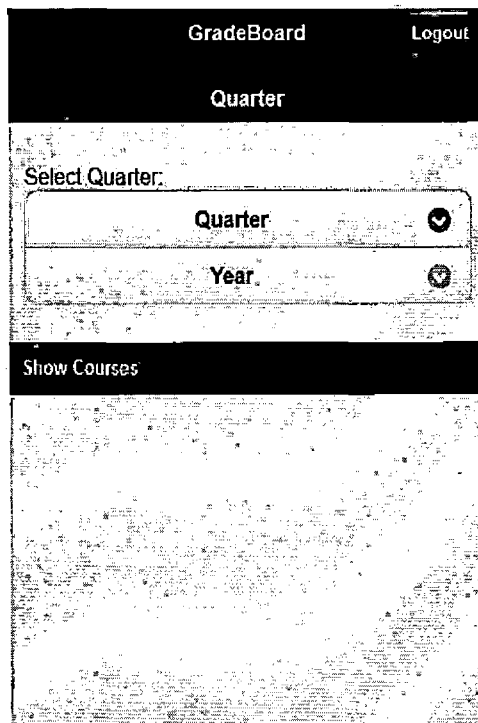


Fig. 4.3: Quarter Screen

4.4 Courses Screen

The courses screen shows a list of courses as shown in Figure 4.4. Using the courses screen, the instructor can add a new course or view the details of the existing course. After the user clicks on the course, the course details page is loaded.

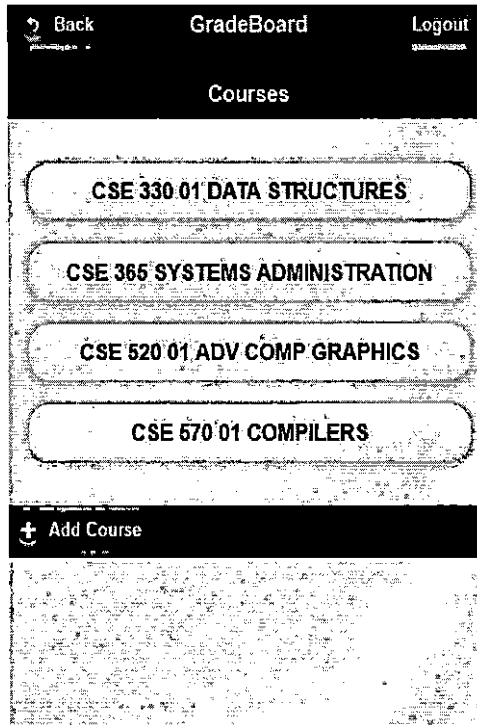


Fig. 4.4: Courses Screen

4.5 Add Course Screen

The add course screen is shown when the user clicks on the add button in the courses screen. Using this screen, instructors can add a new course as shown in Figure 4.5. This page checks that the course value entered already exists and displays an appropriate error message if not. The back button allows users to navigate to the course list page.

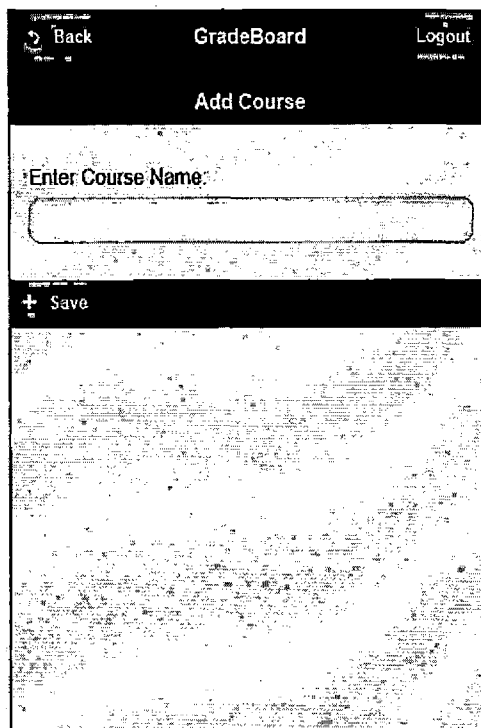


Fig. 4.5: Add Course Screen

4.6 Course Details Screen

The course details screen allows instructors to view course details such as the number of students registered and the number of seats available. The course details screen is shown in Figure 4.6. From the course details screen, the user can edit the course, edit the grade components, or delete the course.

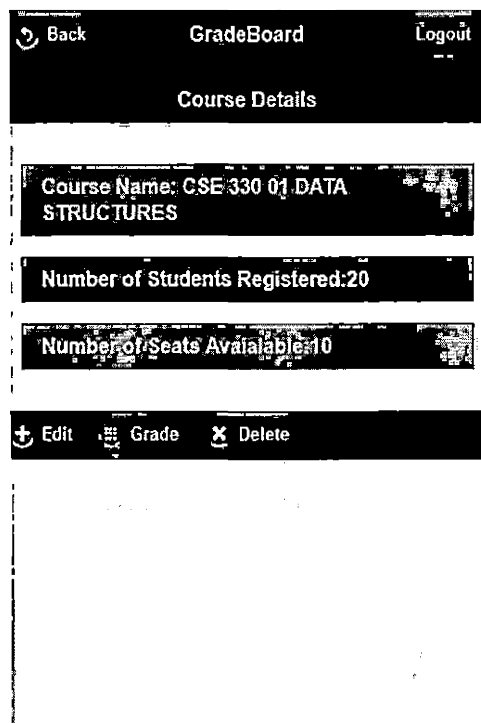


Fig. 4.6: Course Details Screen

4.7 Edit Course Screen

The edit course screen allows instructors to edit the course details as shown in Figure 4.7. The instructor can modify the course name, add additional instructors, add students and add gradable components.

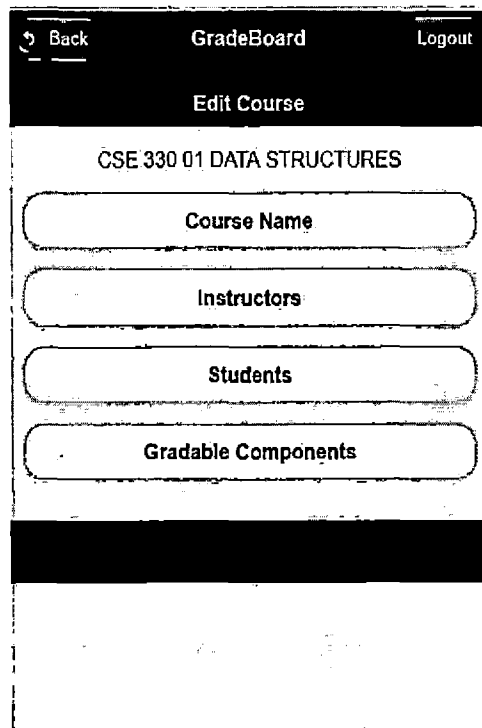


Fig. 4.7: Edit Course Screen

4.8 Edit Course Name Screen

The edit course name screen allows instructors to edit the course name as shown in Figure 4.8. A validation is added to check if the course name entered by the user is empty or if it already exists. Changing the course name involves passing the current course name and new course name to the server. The server searches first the course entity with name equal to the current course name and then replaces the course name property value with the new course name.

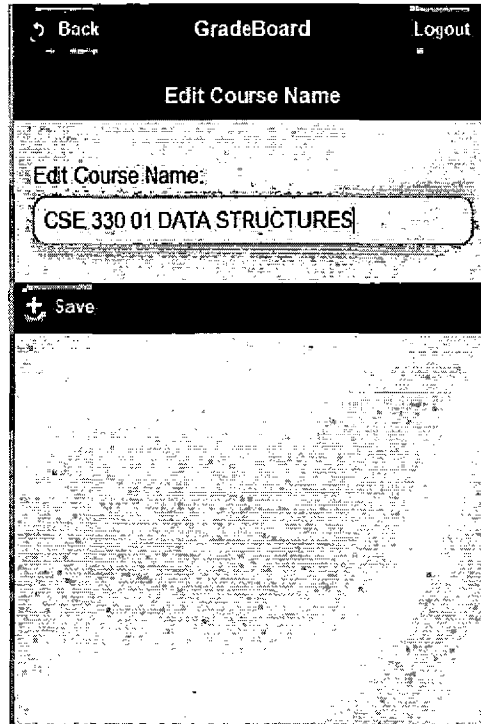


Fig. 4.8: Edit Course Name Screen.

4.9 Instructors Screen

The instructors screen displays a list of instructors for a course as shown in Figure 4.9. A course can have multiple instructors. Internally, instructors are stored as auth objects with a parent instructor (who created the course). Each auth object will contain the id of the course entity.

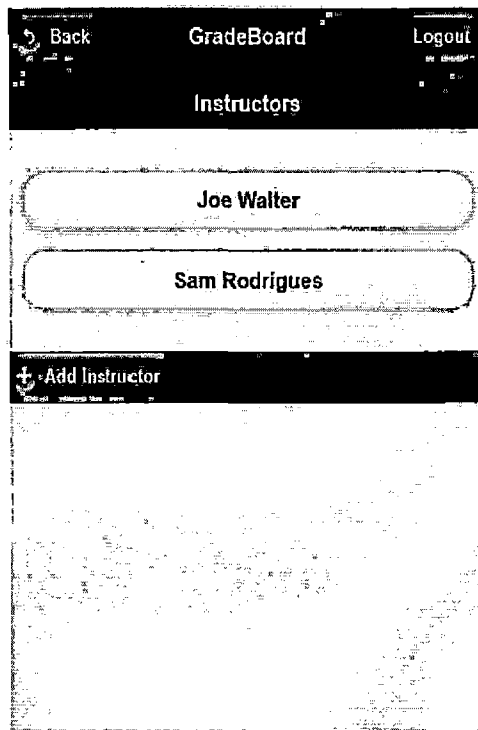


Fig. 4.9: Instructors Screen

4.10 Add Instructor

The add instructor screen allows an instructor to add another instructor as shown in Figure 4.10. Only the instructor who originally created the course can add additional instructors. The new instructors have restricted access. Permission to change the course name or gradable components is not given to new instructors.

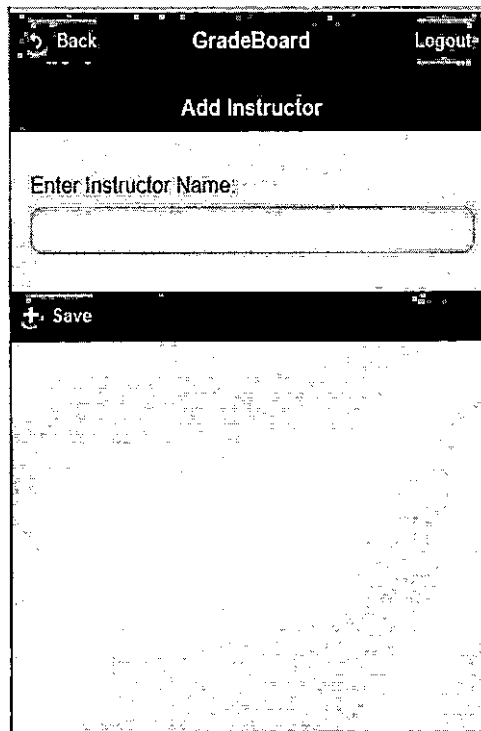


Fig. 4.10: Add Instructor Screen

4.11 Delete Instructor

The delete instructor screen allows the original instructor to remove an instructor from the course as shown in Figure 4.11. The instructor who originally created the course can not be deleted, so there should always be at least one instructor for the course.

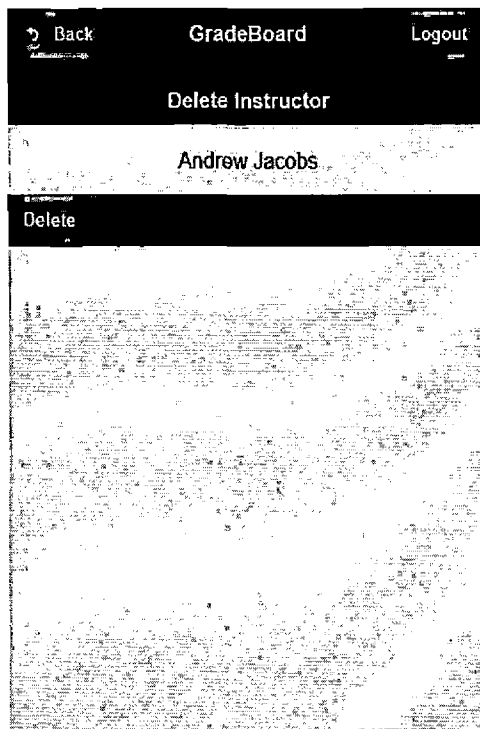


Fig. 4.11: Delete Instructor Screen

4.12 Students

The student screen lists all the students registered for the course as shown in Figure 4.12. New students can be added by clicking on the add student button. There is also an option to add multiple students at once.

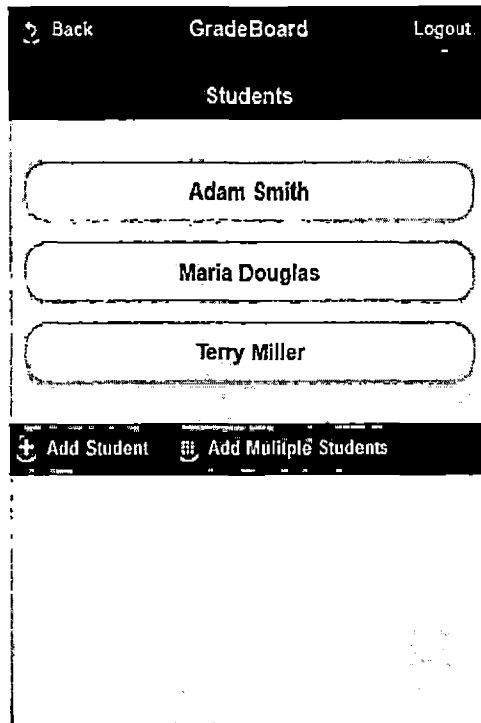
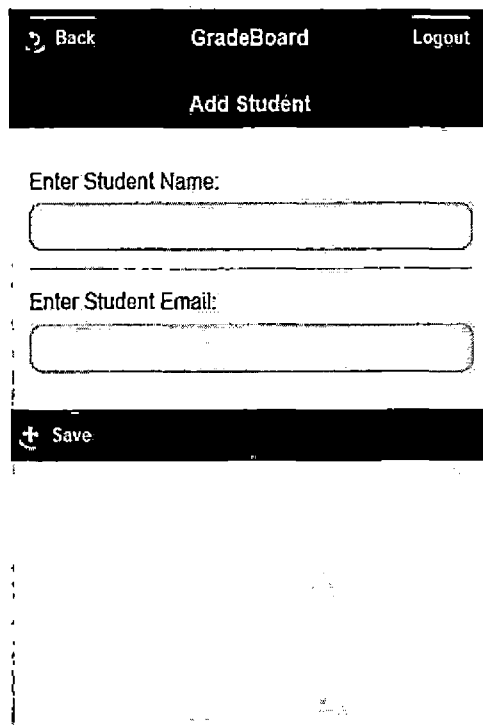


Fig. 4.12: Students Screen

4.13 Add Student

This screen allows instructors to add students to the course as shown in Figure 4.13. A validation is added to check for an empty student name or email address. Also, if a student with same name and email address already exists, an error message is shown to the instructor.



The screenshot shows a mobile application interface for adding a student. At the top, there is a dark navigation bar with three options: 'Back' (with a left arrow icon), 'GradeBoard', and 'Logout'. Below this bar, the title 'Add Student' is centered. The main content area contains two text input fields. The first is labeled 'Enter Student Name:' and the second is labeled 'Enter Student Email:'. At the bottom of the screen, there is a dark bar with a plus icon and the text 'Save'.

Fig. 4.13: Add Student Screen

4.14 Add Multiple Students

The add multiple students screen allows instructors to enter multiple student data at once as shown in Figure 4.14. This helps the instructor to update student data at once when a course is created. To add multiple student data, instructors need to provide a comma separated list of student data in an HTML textarea field. The student data must contain three fields: first name, last name and email address. The instructor is notified if any student record is missing one of these fields.

Back GradeBoard Logout

Add Multiple Students

Input student data in comma seperated list of values(csv format): Each student record should have LastName, FirstName and Email address:

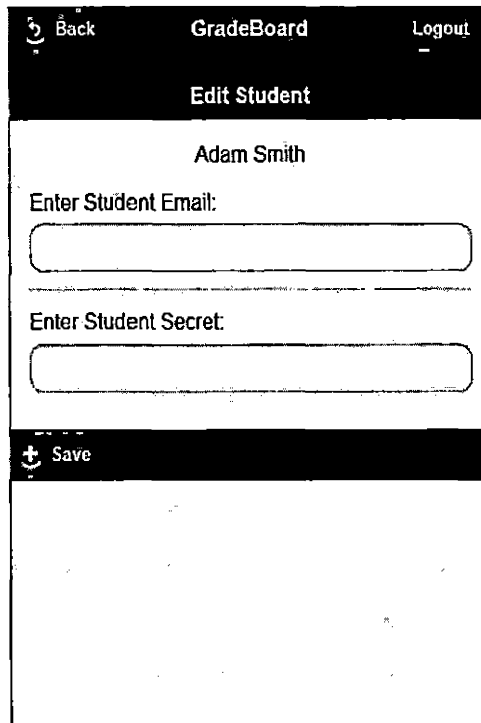
+

Save

Fig. 4.14: Add Multiple Students Screen

4.15 Edit Student

The edit student screen allows instructors to edit the details of a student as shown in Figure 4.15. All of the student information can be changed by the instructor.



The screenshot shows a mobile application interface for editing a student. At the top, there is a dark header bar with three items: a back arrow icon followed by the text 'Back', the application name 'GradeBoard' in the center, and the text 'Logout' on the right. Below the header, the title 'Edit Student' is centered. Underneath the title, the name 'Adam Smith' is displayed. There are two input fields: the first is labeled 'Enter Student Email:' and the second is labeled 'Enter Student Secret:'. At the bottom of the screen, there is a dark bar with a plus sign icon and the text 'Save'.

Fig. 4.15: Edit Student Screen

4.16 Grades

The grades screen shows an instructors list of the names of all gradable components as shown in Figure 4.16. If a new grade is required, then the instructor needs to add the gradable component in the add gradable component screen. Clicking on the grade button lists all the students registered for the course.

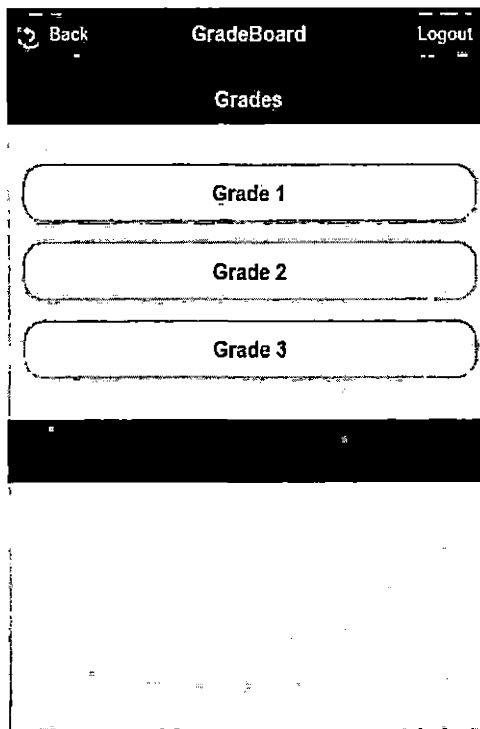


Fig. 4.16: Grades Screen

4.17 Edit Grades

The edit grade screen presents to the instructor a list of students whose grades can be edited as shown in Figure 4.17. Instructors can edit the individual grade by clicking on the student button. The view grade sheet option allows the instructor to view the grade sheet of the entire course.

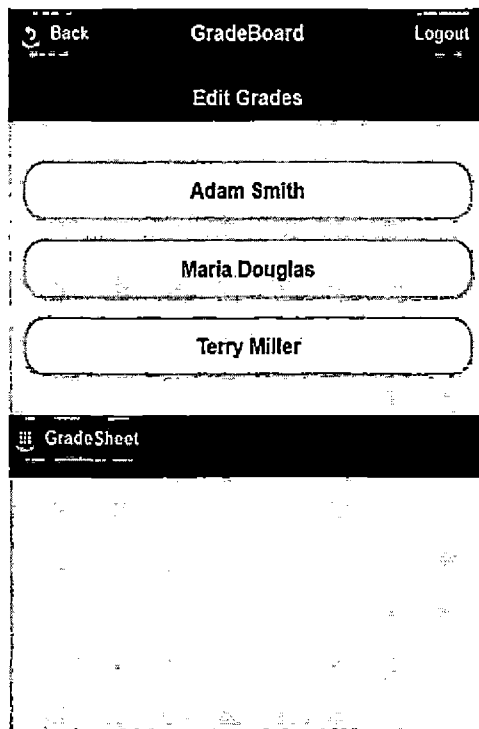
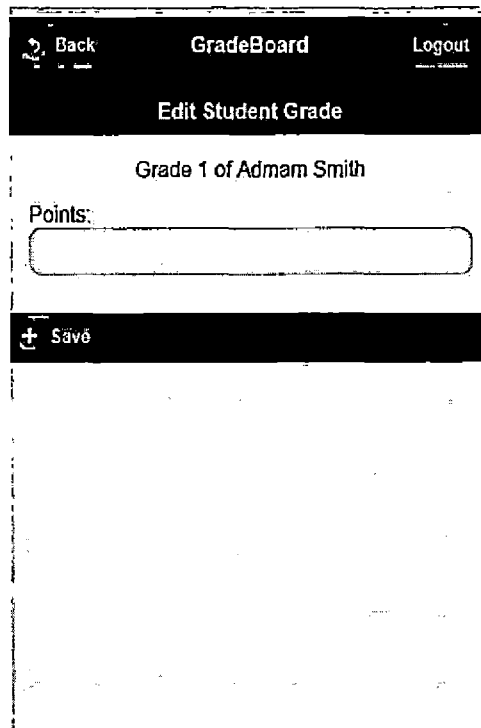


Fig. 4.17: Edit Grades Screen

4.18 Edit Student Grade

The edit student grade screen allows instructors to enter grade points of a student as shown in Figure 4.18. A validation is provided to check that values are not empty and do not exceed the maximum value of grade points entered in the gradable component.



The screenshot shows a mobile application interface for editing a student's grade. At the top, a dark navigation bar contains three items: a left-pointing arrow labeled 'Back', the text 'GradeBoard' in the center, and the text 'Logout' on the right. Below this bar, the title 'Edit Student Grade' is centered. Underneath the title, the text 'Grade 1 of Admam Smith' is displayed. A label 'Points:' is positioned to the left of a rectangular text input field. At the bottom of the screen, there is a dark bar with a plus sign icon on the left and the word 'Save' on the right.

Fig. 4.18: Edit Student Grade Screen

4.19 Gradable Components

The gradable component screen displays a list of gradable components of a course as shown in Figure 4.19. A new gradable component can be added by clicking on the add gradable component button.

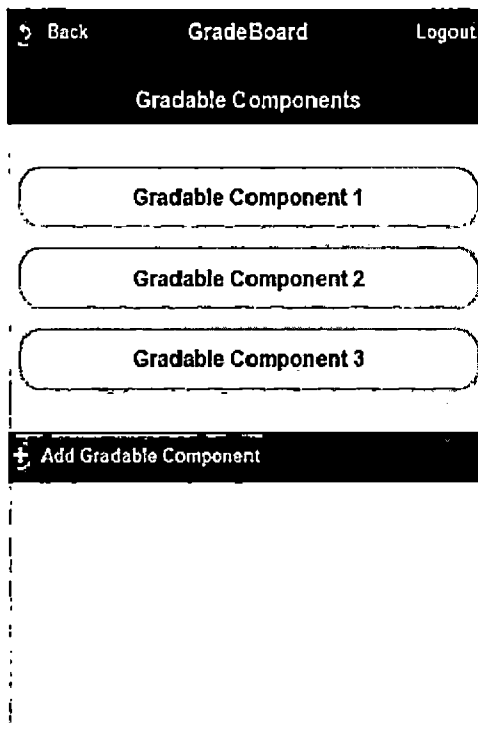


Fig. 4.19: Gradable Component Screen

4.20 Add Gradable Component

The add gradable component screen allows instructors to add a gradable component for the course as shown in Figure 4.20. Name, points and deadline are required fields and a validation is added to check for empty values. A check for duplicated entries is added to make sure gradable components are unique in a course.

The screenshot shows a mobile application interface for adding a gradable component. The top navigation bar is dark with three items: a back arrow labeled 'Back', the title 'GradeBoard', and a 'Logout' button. Below this is a white title bar with the text 'Add Gradable Component'. The main content area is white and contains three vertically stacked input fields. The first is labeled 'Name:', the second 'Points:', and the third 'Deadline:'. Each field has a rounded rectangular border. Below the input fields is a dark bar with a white plus icon and the text 'Add'. At the bottom of the screen, there is a list of existing components. The list has three columns: 'Name', 'Points', and 'Deadline'. The list is currently empty.

Fig. 4.20: Add Gradable Component Screen

4.21 Edit Gradable Component

The edit_gradable component screen allows instructors to edit gradable components for the course as shown in Figure 4.21. Name, points and deadline are required fields, so a validation is performed to check for empty values.

The screenshot shows a mobile application interface for editing a gradable component. At the top, there is a black navigation bar with three items: a back arrow labeled 'Back', the title 'GradeBoard', and a 'Logout' button. Below this bar, the main title 'Edit Gradable Component' is centered. Underneath, the specific component being edited is identified as 'Gradable Component 1'. The form contains three required input fields: 'Name:', 'Points:', and 'Deadline:'. Each field is represented by a rounded rectangular text box. At the bottom of the screen, there is a black bar with a white plus icon and the text 'Save', indicating the action to save the changes.

Fig. 4.21: Edit Gradable Component Screen

5. CONCLUSION AND FUTURE DIRECTION

5.1 *Conclusion*

The GradeBoard application is developed on the Google App Engine cloud platform and has a mobile user interface. By combining Java Servlet technology and using the auto scaling and load balancing features of the cloud platform, the application is robust and built to scale to support large numbers of users. The application is very convenient for instructors and students to access because it works well on mobile devices. The use of Javascript technology on the client side gives the advantage of building one application that runs on multiple platforms. Using the Bigtable data store, attributes of an entity can be dynamically expanded. For example in the GradeBoard application, if there is a requirement to add a new visiting instructor from a different university, a new attribute named university can be added to the instructor entity without changing the structure of the datastore. The GradeBoard application provides public grade sheets to students who can view their performance and improve their grades during different phases of the course. Overall, the GradeBoard application provides scalability, convenient user experience and easier maintenance of the system.

5.2 *Future Direction*

The GradeBoard application can be used as a reference for providing cloud-based solutions and mobile user interfaces. The GradeBoard application can be further

extended to deploy to other cloud platforms such as Amazon Web Services, Heroku, Microsoft Azure, etc. The following is a list of the possible future enhancements to the application.

- Provide a responsive design using technologies such as Bootstrap to support dynamic screen sizes on multiple devices and desktop systems [5].
- Extend gradable components to incorporate game-like features.
- Add new interfaces to the data model to support alternative database systems that could be used at lower cost, such as MongoDB [1].

APPENDIX A
SERVER SOURCE CODE

```

//Auth.java

package gradesys;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.google.appengine.api.datastore.Cursor;
import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;
import com.google.appengine.api.datastore.Entity;
import com.google.appengine.api.datastore.EntityNotFoundException;
import com.google.appengine.api.datastore.Key;
import com.google.appengine.api.datastore.Query;
import com.google.appengine.api.datastore.QueryResultIterator;
import com.google.appengine.api.datastore.QueryResultList;
import com.google.appengine.api.datastore.Transaction;
import com.google.appengine.api.datastore.Query.FilterOperator;
import com.google.appengine.api.datastore.Query.FilterPredicate;

public class Auth {

    private static final String entityKind = "auth";
    private static final String namePropertyName = "courseId";
    private Entity entity = null;

    private Auth(Entity entity) {
        this.entity = entity;
    }
}

```



```

public Long getID() {
    return (Long) entity.getKey().getId();
}

public Long getCourseId() {
    return (Long) entity.getProperty(namePropertyName);
}

public Key getInstructor() {
    return entity.getParent();
}

private void setName(Long courseId) {
    entity.setProperty(namePropertyName, courseId);
}

public void save() {
    saveOrCreateEntity(entity);
}

public static void deleteByName(Long courseId) {
    deleteEntityByName(courseId);
}

public static Auth create(Long courseId, Key instructorKey)
    throws CourseAlreadyExistsException {
    return new Auth(createEntity(courseId, instructorKey));
}

```

```

public static Auth getByCourseId(Long courseId) {
    Entity entity = getEntityByCourseId(courseId);
    if (entity == null) {
        return null;
    } else {
        return new Auth(entity);
    }
}

// Helper function that runs inside or outside a transaction.
private static Entity getEntityByCourseId(Long courseId) {
    Query query = new Query(entityKind);
    Query.Filter filter = new FilterPredicate(namePropertyName,
        FilterOperator.EQUAL, courseId);
    query.setFilter(filter);
    DatastoreService datastore = DatastoreServiceFactory
        .getDatastoreService();
    return datastore.prepare(query).asSingleEntity();
}

public static List<Course> getCoursesByInstructor(Key instructorKey)
    throws EntityNotFoundException {
    Query query = new Query(entityKind);
    query.setAncestor(instructorKey);

    DatastoreService datastore = DatastoreServiceFactory
        .getDatastoreService();

```

```

Iterator<Entity> iterator = datastore.prepare(query).asIterator
    ();
List<Course> courses = new ArrayList<Course>();
while (iterator.hasNext()) {
    Entity entity = iterator.next();
    Long courseId = (Long) entity.getProperty(
        namePropertyName);
    Course course = Course.getByCourseId(courseId);
    courses.add(course);
}
return courses;
}

```

```

public static Course getCourseDetailsByInstructor(String id,
    Key instructorKey) throws EntityNotFoundException {
    Query query = new Query(entityKind);
    query.setAncestor(instructorKey);
    Query.Filter filter = new FilterPredicate(namePropertyName,
        FilterOperator.EQUAL, id);
    query.setFilter(filter);
    DatastoreService datastore = DatastoreServiceFactory
        .getDatastoreService();
    Entity instructorEntity = datastore.prepare(query).
        asSingleEntity();
    Long courseId = (Long) instructorEntity.getProperty(
        namePropertyName);
    Course course = Course.getByCourseId(courseId);
    return course;
}

```

```
)
```

```
// Helper function that runs inside or outside a transaction.
```

```
private static void saveOrCreateEntity(Entity entity) {  
    DatastoreService datastore = DatastoreServiceFactory  
        .getDatastoreService();  
    datastore.put(entity);  
}
```

```
private static void deleteEntityByName(Long courseId) {  
    DatastoreService datastore = DatastoreServiceFactory  
        .getDatastoreService();  
    Transaction txn = datastore.beginTransaction();  
    Entity entity = getEntityByCourseId(courseId);  
    if (entity != null) {  
        datastore.delete(entity.getKey());  
    }  
    txn.commit();  
}
```

```
private static Entity createEntity(Long courseId, Key instructorKey)  
    throws CourseAlreadyExistsException {  
    DatastoreService datastore = DatastoreServiceFactory  
        .getDatastoreService();  
    Transaction txn = datastore.beginTransaction();  
    Entity entity = getEntityByCourseId(courseId);  
    if (entity != null) {  
        txn.commit();  
        throw new CourseAlreadyExistsException();  
    }  
}
```

```

        )
        entity = new Entity(entityKind, instructorKey);
        entity.setProperty(namePropertyName, courseId);
        datastore.put(entity);

        txn.commit();

        return entity;
    }
}

```

```
//Course.java
```

```

package gradesys;

import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;
import com.google.appengine.api.datastore.Entity;
import com.google.appengine.api.datastore.EntityNotFoundException;
import com.google.appengine.api.datastore.Key;
import com.google.appengine.api.datastore.KeyFactory;
import com.google.appengine.api.datastore.Query;
import com.google.appengine.api.datastore.Query.FilterOperator;
import com.google.appengine.api.datastore.Query.FilterPredicate;
import com.google.appengine.api.datastore.Transaction;

public class Course {

    private static final String entityKind = "course";
    private static final String namePropertyName = "name";

    private Entity entity = null;
}

```

```

private Course(Entity entity) {
    this.entity = entity;
}

public Long getID() {
    return (Long) entity.getKey().getId();
}

public String getName() {
    return (String) entity.getProperty(namePropertyName);
}

private void setName(String name) {
    entity.setProperty(namePropertyName, name);
}

public void save() {
    saveOrCreateEntity(entity);
}

public static void deleteByName(String name) {
    deleteEntityByName(name);
}

public static Course create(String name)
    throws CourseAlreadyExistsException {
    return new Course(createEntity(name));
}

```

```

public static Course save(String oldCourseName, String newCourseName)
    throws Exception {
    Course course = getByName(oldCourseName);
    if (course == null) {
        throw new CourseNotFoundException();
    } else {
        DatastoreService datastore = DatastoreServiceFactory
            .getDatastoreService();
        Transaction txn = datastore.beginTransaction();
        course.setName(newCourseName);
        datastore.put(course.entity);
        txn.commit();
        return course;
    }
}

```

```

public static Course getByName(String name) {
    Entity entity = getEntityByName(name);
    if (entity == null) {
        return null;
    } else {
        return new Course(entity);
    }
}

```

```

public static Course getByCourseId(Long courseId)
    throws EntityNotFoundException {
    Entity entity = getEntityByKey(courseId);
    if (entity == null) {

```

```

        return null;
    } else {
        return new Course(entity);
    }
}

// Helper function that runs inside or outside a transaction.
public static Entity getEntityByKey(Long id) throws
    EntityNotFoundException {
    /*
     * Query query = new Query(entityKind); Query.Filter filter =
        new
     * FilterPredicate(Entity.KEY_RESERVED_PROPERTY, FilterOperator.
        EQUAL,
     * id); query.setFilter(filter);
    */
    Key key = KeyFactory.createKey("course", id);

    DatastoreService datastore = DatastoreServiceFactory
        .getDatastoreService();
    return datastore.get(key);
}

// Helper function that runs inside or outside a transaction.
private static Entity getEntityByName(String name) {
    Query query = new Query(entityKind);
    Query.Filter filter = new FilterPredicate(namePropertyName,
        FilterOperator.EQUAL, name);
    query.setFilter(filter);
}

```



```

        DatastoreService datastore = DatastoreServiceFactory
            .getDatastoreService();
        return datastore.prepare(query).asSingleEntity();
    }

    // Helper function that runs inside or outside a transaction.
    private static void saveOrCreateEntity(Entity entity) {
        DatastoreService datastore = DatastoreServiceFactory
            .getDatastoreService();
        datastore.put(entity);
    }

    private static void deleteEntityByName(String name) {
        DatastoreService datastore = DatastoreServiceFactory
            .getDatastoreService();
        Transaction txn = datastore.beginTransaction();
        Entity entity = getEntityByName(name);
        if (entity != null) {
            datastore.delete(entity.getKey());
        }
        txn.commit();
    }

    private static Entity createEntity(String name)
        throws CourseAlreadyExistsException {
        DatastoreService datastore = DatastoreServiceFactory
            .getDatastoreService();
        Transaction txn = datastore.beginTransaction();
        Entity entity = getEntityByName(name);

```

```

        if (entity != null) {
            txn.commit();
            throw new CourseAlreadyExistsException();
        }
        entity = new Entity(entityKind);
        entity.setProperty(namePropertyName, name);
        datastore.put(entity);
        txn.commit();
        return entity;
    }
}

```

//Instructor.java:

```

package gradesys;

import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;
import com.google.appengine.api.datastore.Entity;
import com.google.appengine.api.datastore.Key;
import com.google.appengine.api.datastore.Query;
import com.google.appengine.api.datastore.Transaction;
import com.google.appengine.api.datastore.Query.FilterOperator;
import com.google.appengine.api.datastore.Query.FilterPredicate;

public class Instructor {

    private static final String entityKind = "Instructor";

    private static final String namePropertyName = "userId";

```

```
private Entity entity = null;

private Instructor(Entity entity) {
    this.entity = entity;
}

public Long getID() {
    return (Long) entity.getKey().getId();
}

public Key getKey() {
    return entity.getKey();
}

public String getName() {
    return (String) entity.getProperty(namePropertyName);
}

private void setName(String name) {
    entity.setProperty(namePropertyName, name);
}

public void save() {
    saveOrCreateEntity(entity);
}

public static void deleteByName(String name) {
    deleteEntityByName(name);
}
```

```

}

public static Instructor create(String name) throws
    CourseAlreadyExistsException {
    return new Instructor(createEntity(name));
}

public static Instructor getByName(String name) {
    Entity entity = getEntityByName(name);
    if (entity == null) {
        return null;
    } else {
        return new Instructor(entity);
    }
}

// Helper function that runs inside or outside a transaction.
private static Entity getEntityByName(String name) {
    Query query = new Query(entityKind);
    Query.Filter filter = new FilterPredicate(namePropertyName,
        FilterOperator.EQUAL, name);
    query.setFilter(filter);
    DatastoreService datastore = DatastoreServiceFactory.
        getDatastoreService();
    return datastore.prepare(query).asSingleEntity();
}

// Helper function that runs inside or outside a transaction.
private static void saveOrCreateEntity(Entity entity) {

```

```

        DatastoreService datastore = DatastoreServiceFactory.
                                getDatastoreService();

        datastore.put(entity);
    }

private static void deleteEntityByName(String name) {
    DatastoreService datastore = DatastoreServiceFactory.
                                getDatastoreService();

    Transaction txn = datastore.beginTransaction();
    Entity entity = getEntityByName(name);
    if (entity != null) {
        datastore.delete(entity.getKey());
    }
    txn.commit();
}

private static Entity createEntity(String name) throws
    CourseAlreadyExistsException {
    DatastoreService datastore = DatastoreServiceFactory.
                                getDatastoreService();

    Transaction txn = datastore.beginTransaction();
    Entity entity = getEntityByName(name);
    if (entity != null) {
        txn.commit();
        throw new CourseAlreadyExistsException();
    }
    entity = new Entity(entityKind);
    entity.setProperty(namePropertyName, name);
    datastore.put(entity);
}

```

```
        txn.commit();

        return entity;
    }
}
```

\\InstructorControllerServlet.java:

```
package gradesys;

import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.appengine.api.datastore.EntityNotFoundException;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;
import com.google.appengine.labs.repackaged.org.json.JSONArray;
import com.google.appengine.labs.repackaged.org.json.JSONObject;

@SuppressWarnings("serial")
public class InstructorControllerServlet extends HttpServlet {
```

```

private static final Logger logger = Logger.
    getLogger(InstructorControllerServlet.class.getName());

private String addCourse(HttpServletRequest req,
    HttpServletResponse resp) {
    String courseName = req.getParameter("name");
    try {
        Course course = Course.create(courseName);
        UserService userService = UserServiceFactory.
            getUserService();
        User user = userService.getCurrentUser();
        Instructor instructor = Instructor.
            getByName(user.getUserId());
        if(instructor == null) {
            instructor = Instructor.create(user.getUserId())
                ;
            Auth.create(course.getID(), instructor.getKey())
                ;
        } else {
            Auth.create(course.getID(), instructor.getKey())
                ;
        }
        List<Course> courses = Auth.getCoursesByInstructor
            (instructor.getKey());
        return Util.getCoursesJson(courses);
    } catch (CourseAlreadyExistsException e) {
        return "{ \"error\": \"Course name already taken.\" }";
    } catch (EntityNotFoundException e) {

```

```

        logger.log(Level.SEVERE, "Error getting courses", e);
        return "{}";
    } catch (Exception e) {
        logger.log(Level.SEVERE, "Error adding course", e);
        return "{}";
    }
}

private String saveCourse(HttpServletRequest req,
                        HttpServletResponse resp) {
    String oldCourseName = req.getParameter("oldCourseName");
    String newCourseName = req.getParameter("newCourseName");
    if(Util.isEmpty(oldCourseName) || Util.isEmpty(newCourseName)) {
        return "{}";
    }
    try {
        UserService userService = UserServiceFactory.
            getUserService();
        User user = userService.getCurrentUser();
        Instructor instructor = Instructor.getByName(user.
            getUserId());
        if(instructor == null) {
            return "{}";
        }
        Course.save(oldCourseName, newCourseName);
        List<Course> courses = Auth.
            getCoursesByInstructor(instructor.getKey
                ());
        return Util.getCoursesJson(courses);
    }
}

```



```

    } catch (CourseNotFoundException e) {
        logger.log(Level.SEVERE, "Error getting courses", e);
        return "{ \"error\": \"Course that you are saving does
            not exists.\" }";
    } catch (EntityNotFoundException e) {
        logger.log(Level.SEVERE, "Error getting courses", e);
        return "{ \"error\": \"Course that you are saving does
            not exists.\" }";
    } catch (Exception e) {
        logger.log(Level.SEVERE, "Error saving course", e);
        return "{}";
    }
}
}

```

```

private String listCourses(HttpServletRequest req,
    HttpServletResponse resp) {
    try {
        UserService userService = UserServiceFactory.
            getUserService();
        User user = userService.getCurrentUser();
        Instructor instructor = Instructor.getByName(user.
            getUserId());
        if(instructor == null) {
            return "{}";
        }
        List<Course> courses = Auth.getCoursesByInstructor
            (instructor.getKey());
        if(courses.size() == 0)
            return "{}";
    }
}

```

```

JSONArray jsonArray = new JSONArray();
for(int i = 0; i < courses.size(); i++) {
    JSONObject jsonObject = new JSONObject();
    Course course = courses.get(i);
    jsonObject.put("name", course.getName());
    jsonObject.put("id", course.getID());
    jsonArray.put(jsonObject);
}
JSONObject jsonObject = new JSONObject();
jsonObject.put("courses", jsonArray);
return jsonObject.toString();
} catch (Exception e) {
    return "{ \"err\": \"Unable to list courses.\" }";
}
}

```

```

private String listCourse(HttpServletRequest req,
    HttpServletResponse resp) {
    try {
        String courseId = req.getParameter("id");
        if(Util.isEmpty(courseId)) {
            return "{}";
        }
        UserService userService = UserServiceFactory.
            getUserService();
        User user = userService.getCurrentUser();
        Instructor instructor = Instructor.getByName(user.
            getUserId());
    }
}

```

```

        if(instructor == null) {
            return "{}";
        }

        Course course = Auth.getCourseDetailsByInstructor(courseId, instructor.getKey());

        if(course == null)
            return "{}";

        JSONObject jsonObject = new JSONObject();
        jsonObject.put("courseName", course.getName());
        return jsonObject.toString();
    } catch (Exception e) {
        return "{ \"err\": \"Unable to list courses.\" }";
    }
}

```

```

@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    String operation = req.getParameter("op");
    if(operation != null) { // !Util.isEmpty(operation) }
        if(operation.equalsIgnoreCase("courseadd")) {
            resp.getWriter().write(addCourse(req, resp));
        }
        if(operation.equalsIgnoreCase("coursesave")) {
            resp.getWriter().write(saveCourse(req, resp));
        }
        if(operation.equalsIgnoreCase("listcourse")) {

```

```

        resp.getWriter().write(listCourse(req, resp));
    }
} else {
    resp.getWriter().print("{ \"err\": \"unknown operation\"
        }");
}
}
}

```

```

@Override
protected void doGet(HttpServletRequest req,
    HttpServletResponse resp)
    throws ServletException, IOException {
    String operation = req.getParameter("op");
    if(operation != null) { // !Util.isEmpty(operation) {
        if(operation.equalsIgnoreCase("listCourses")) {
            resp.getWriter().write(listCourses(req, resp));
        }
    } else {
        resp.getWriter().print("{ \"err\": \"unknown operation\"
            }");
    }
}
}
}

```

```

}

```

```

\\LoginFilter.java:

```

```

package gradesys;

```

```

import java.io.IOException;

```

```

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

public class LoginFilter implements Filter {

    @Override
    public void destroy() {
        // TODO Auto-generated method stub
    }

    @Override
    public void doFilter(ServletRequest req, ServletResponse resp,
        FilterChain chain) throws IOException, ServletException
    {

        HttpServletRequest httpReq = (HttpServletRequest) req;
        HttpServletResponse httpResp = (HttpServletResponse) resp;

```

```

UserService userService = UserServiceFactory.getUserService();
User user = userService.getCurrentUser();
if(user == null) {
    httpResp.sendRedirect(userService.
        createLoginURL(httpReq.getServletPath()));
} else {
    String csrfToken = CsrfCipher.encryptUserId(user.
        getUserId());
    httpReq.setAttribute("csrfToken", csrfToken);
    httpReq.setAttribute("user", user);
    chain.doFilter(req, resp);
}
}

@Override
public void init(FilterConfig arg0) throws ServletException {
    // TODO Auto-generated method stub
}
}

```

// LogoutFilter.java:

```

package gradesys;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;

```

```
import javax.servlet.http.HttpServletResponse;

import com.google.appengine.api.users.UserServiceFactory;

@SuppressWarnings("serial")
public class LogoutServlet extends HttpServlet {

    @Override
    protected void doGet (HttpServletRequest req,
                          HttpServletResponse resp)
        throws ServletException, IOException {
        resp.sendRedirect (UserServiceFactory.
                          getUserService().createLogoutURL(
                              "/instructor"));
    }
}
```

APPENDIX B
CLIENT SOURCE CODE


```

// app.js

var app = {
    controller : {},
    view : {},
    model : {},
    util : {}
};

/** App view for all the classes */

(function() {
    var courses;

    app.view.transfer = function(toPage) {
        $.mobile.changePage(toPage,
            {
                allowSamePageTransition : true,
                transition : 'none',
                reloadPage : false
            });
    }
})();

/** Course List */
app.model.courseList = {};

(function() {

```

```

var courses;

app.model.courseList.update = function(data) {
    courses = data;
};

app.model.courseList.get = function() {
    return courses;
};
})();

app.view.courseList = {};

(function(mcourseList) {

    var courses;

    app.view.courseList.render = function() {

        //render : new function() {
        if(courses === null && mcourseList.get() === null)
            return;

        if (courses === mcourseList.get())
            return;

        courses = mcourseList.get();
        $("#main > p").remove();
        $.each(courses.courses, function(i, course) {

```

```

    var $p = $('<p></p>');
    var query = "CourseOperations.html"; //?courseName=" + course;
    var onclick = "app.model.courseOperations.setName(' " + course.name +
        "');app.model.courseOperations.setId(" + course.id + ");app.
        view.transfer(' " + query + "')";
    var $a = $('<a data-role="button" data-transition="slide" onclick="'
        + onclick + ' " href="' + query + '></a>');
    $('#main').append($p);
    $a.html(course.name);
    $p.append($a);
    });
    $('#main').trigger('create');
};

)) (app.model.courseList);

app.controller.courseList = {};

(function(jQuery, mcourseList, vcourseList) {

    app.controller.courseList.init = function() {

        // render : new function() {
        //mcourseList.update(vcourseList.render);
        $.ajax({
            type : 'GET',
            url : "/instructor/controller",
            data : {

```

```

        op : 'listCourses'
    },
    dataType : "json"
  }).done(function(data) {
    if (data.err) {
      alert(data.err);
    } else {
      mcourseList.update(data);
      vcourseList.render();
    }
  }).fail(function(jqXHR, textStatus) {
    console.log(textStatus);
    $.mobile.hidePageLoadingMsg();
  });

  $('#courseList').die().live('pagechange', vcourseList.render);

  //});
});

})(jQuery, app.model.courseList, app.view.courseList);

/***** Course Operations *****/
app.model.courseOperations = {};

(function() {
  var name;

```

```

var id;
var courseDetails;

app.model.courseOperations.setId = function(data) {
    id = data
};

app.model.courseOperations.getName = function() {
    return name;
};

app.model.courseOperations.setName = function(data) {
    name = data;
};

app.model.courseOperations.getId = function() {
    return id;
};

app.model.courseOperations.setCourseDetails = function(data) {
    courseDetails = data;
};

app.model.courseOperations.getCourseDetails = function() {
    return courseDetails;
};
})();

app.view.courseOperations = {};

```

```

(function() {

    app.view.courseOperations.render = function(coursename) {

        $('#copContent').empty();
        $("#copContent > p").remove();
        var courseDetails = app.model.courseOperations.getCourseDetails();

        if(courseDetails === null || courseDetails.courseName === null)
            return;

        var $p1 = $('<p></p>');
        var coursename = "Course Name : " + courseDetails.courseName;
        $p1.append(coursename);
        $('#copContent').append($p1);

        $('#copContent').trigger('create');
    };

})();

app.controller.courseOperations = {};

(function(jQuery, mcourseList, vcourseOperations) {

```

```

app.controller.courseOperations.listCourse = function() {
    var id = app.model.courseOperations.getId();
    $.ajax({
        type : 'POST',
        url : "/instructor/controller",
        data : {
            id : id,
            op : 'listcourse'
        },
        dataType : "json"
    }).done(function(data) {
        if (data.err) {
            console.log(data.err);
        } else {
            app.model.courseOperations.setCourseDetails(data);
            vcourseOperations.render(newCourseName);
        }
    }).fail(function(jqXHR, textStatus) {
        console.log(textStatus);
        $.mobile.hidePageLoadingMsg();
    });
};

```

```

app.controller.courseOperations.saveCourse = function() {
    var newCourseName = $('#courseName').val();
    var oldCourseName = app.model.courseOperations.getName();
    $.ajax({
        type : 'POST',
        url : "/instructor/controller",

```

```

        data : {
            oldCourseName : oldCourseName,
            newCourseName : newCourseName,
            op : 'coursesave'
        },
        dataType : "json"
    }).done(function(data) {
        if (data.err) {
            alert(data.err);
        } else {
            mcourseList.update(data);
            app.model.courseOperations.setCourseName(newCourseName);
            vcourseOperations.render(newCourseName);
        }
    }).fail(function(jqXHR, textStatus) {
        consol.log(textStatus);
        $.mobile.hidePageLoadingMsg();
    });
});

})(jQuery, app.model.courseList, app.view.courseOperations);

/** Course Edit */

app.model.courseEdit = {};

(function() {

```



```

var courseName;

app.model.courseEdit.update = function(data) {
    courseName = data;
};

app.model.courseEdit.get = function() {
    return courseName;
};
})();

app.view.courseEdit = {};

(function() {
    var courses;

    app.view.courseEdit.render = function(coursename) {
        $('#title').empty();
        $('#title').append(coursename);
        $('#courseName').val(coursename);
        $('#header').trigger('create');
    };

})();

app.controller.courseEdit = {};

/***** Utility *****/

```

```

(function() {

    app.util.getUrlVars = function() {

        var vars = [], hash;

        var href = window.location.href;

        var queryUrl = href.slice(href.lastIndexOf( '?' ) + 1);

        var hashes = queryUrl.split( '&' );

        for ( var i = 0; i < hashes.length; i++) {

            hash = hashes[i].split( '=' );

            vars.push(hash[0]);

            vars[hash[0]] = hash[1];

        }

        return vars;

    }

})();

```

```

(function() {

    app.view.displayError = function(error) {

        $('#popupErrorMsg').html(error);

        $('#popupError').popup('open');

    }

})();

```

REFERENCES

- [1] Agile and scalable. <http://www.mongodb.org/>.
- [2] Ajax (programming). [http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming)).
- [3] Amazon Web Services. <http://aws.amazon.com/>.
- [4] Android Mobile Computing Platform.
<http://developer.android.com/about/index.html>.
- [5] Bootstrap. <http://twitter.github.com/bootstrap/>.
- [6] Cloud Computing Document.
http://www.en.wikipedia.org/wiki/Cloud_computing.
- [7] Eclipse Founcation. <http://www.eclipse.org/>.
- [8] Entities, Properties, and Keys.
<https://developers.google.com/appengine/docs/java/datastore/entities>.
- [9] Glossary. <http://technet.microsoft.com/en-us/library/bb742416.aspx>.
- [10] Google Developers Academy.
<https://developers.google.com/appengine/training/intro/whatisgae/>.
- [11] Heroku. <http://www.heroku.com/>.
- [12] How Entities and Indexes are Stored.
https://developers.google.com/appengine/articles/storage_breakdown.
- [13] Http secure. <http://en.wikipedia.org/wiki/HTTPS>.
- [14] Index Definition and Structure.
<https://developers.google.com/appengine/docs/python/datastore/indexes>.

- [15] IOS Mobile Computing Platform. <http://en.wikipedia.org/wiki/IOS>.
- [16] Java Database Connectivity.
<http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc>.
- [17] Jetty (web server). [http://en.wikipedia.org/wiki/Jetty_\(web_server\)](http://en.wikipedia.org/wiki/Jetty_(web_server)).
- [18] JQuery Mobile 1.2 Reference Document.
<http://www.jquerymobile.com/demos/1.2.0/docs/about/features.html>.
- [19] JQuery Official Document. <http://www.jquery.com/>.
- [20] Json. <http://en.wikipedia.org/wiki/Json>.
- [21] Model view controller. <http://en.wikipedia.org/wiki/Model-view-controller>.
- [22] Nosql. <http://en.wikipedia.org/wiki/NoSQL/>.
- [23] Official Documentation of Google App Engine Java Datastore.
<https://developers.google.com/appengine/docs/java/datastore/>.
- [24] Official Documentation of Java Servlet Technology.
<http://www.oracle.com/technetwork/java/whitepaper-135196.html>.
- [25] Official Java Technology Document.
http://www.java.com/en/download/faq/whatis_java.xml.
- [26] PhoneGap. <http://phonegap.com/>.
- [27] PolyModel.
<https://developers.google.com/appengine/docs/python/ndb/polymodelclass>.
- [28] Unified Modeling Language. <http://www-01.ibm.com/software/rational/uml/>.
- [29] W3C community. <http://www.w3.org/TR/REC-html40/>.
- [30] Webopedia Definitions And Terms.
<http://www.webopedia.com/TERM/J/J2EE.html/>.
- [31] What is an Object?
<http://docs.oracle.com/javase/tutorial/java/concepts/object.html>.

[32] Windows Azure. <http://www.windowsazure.com/en-us/develop/overview/>.