Blackboard

CHRISTOPHER LICATA

SHUO PENG

BOJUN SUN

QUN WU

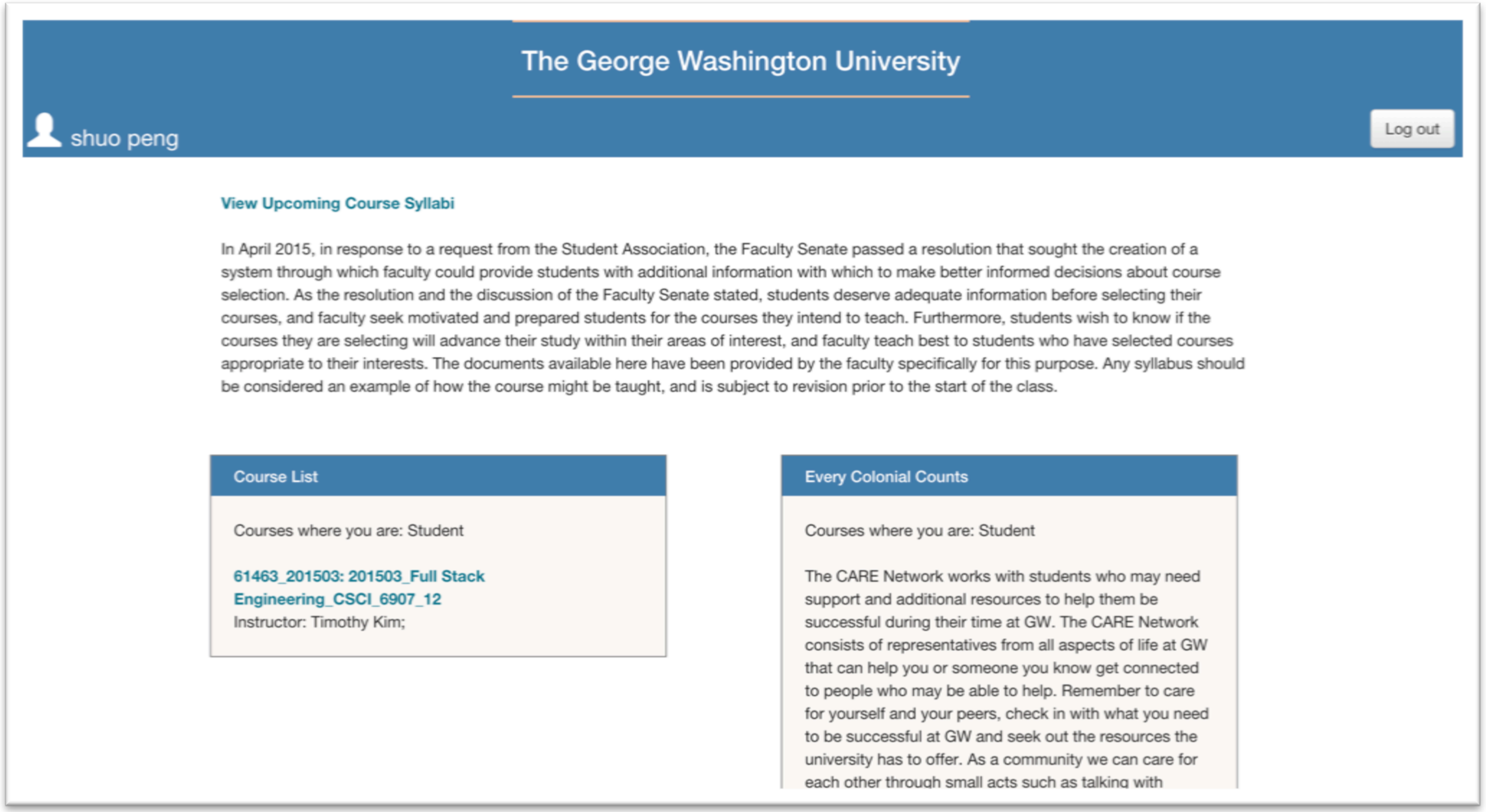
Web Application

BOJUN SUN AND SHUO PENG

Application’s Features

- Sign up/ Log in
- Choose courses

Did the demo present all of those features?



View Upcoming Course Syllabi

In April 2015, in response to a request from the Student Association, the Faculty Senate passed a resolution that sought the creation of a system through which faculty could provide students with additional information with which to make better informed decisions about course selection. As the resolution and the discussion of the Faculty Senate stated, students deserve adequate information before selecting their courses, and faculty seek motivated and prepared students for the courses they intend to teach. Furthermore, students wish to know if the courses they are selecting will advance their study within their areas of interest, and faculty teach best to students who have selected courses appropriate to their interests. The documents available here have been provided by the faculty specifically for this purpose. Any syllabus should be considered an example of how the course might be taught, and is subject to revision prior to the start of the class.

Course List

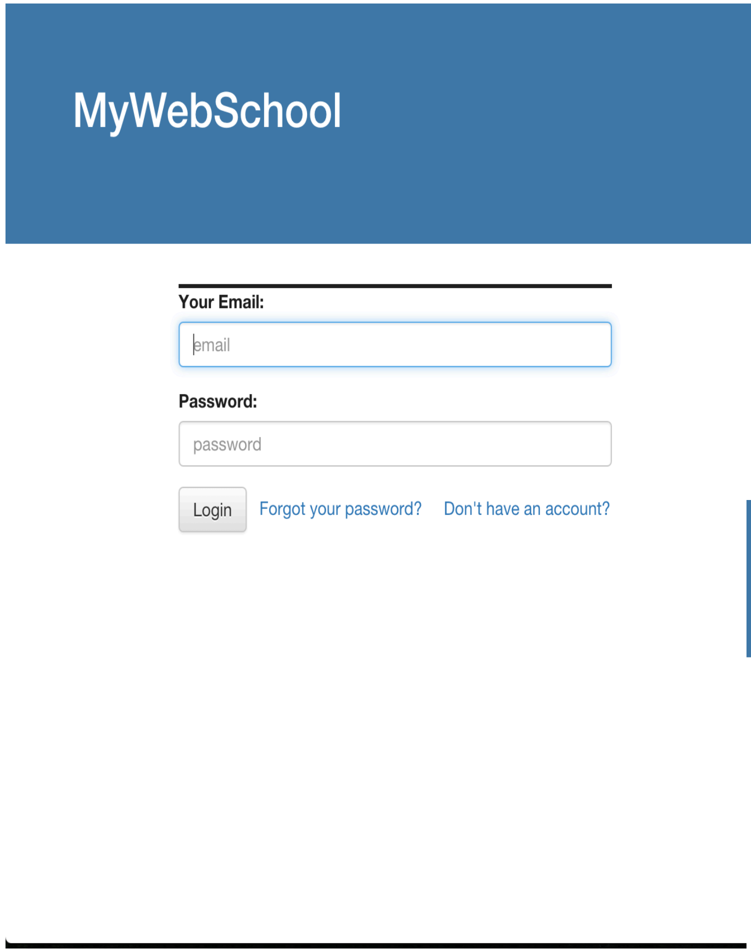
Courses where you are: Student

61463_201503: 201503_Full Stack Engineering_CSCI_6907_12
Instructor: Timothy Kim;

Every Colonial Counts

Courses where you are: Student

The CARE Network works with students who may need support and additional resources to help them be successful during their time at GW. The CARE Network consists of representatives from all aspects of life at GW that can help you or someone you know get connected to people who may be able to help. Remember to care for yourself and your peers, check in with what you need to be successful at GW and seek out the resources the university has to offer. As a community we can care for each other through small acts such as talking with


MyWebSchool

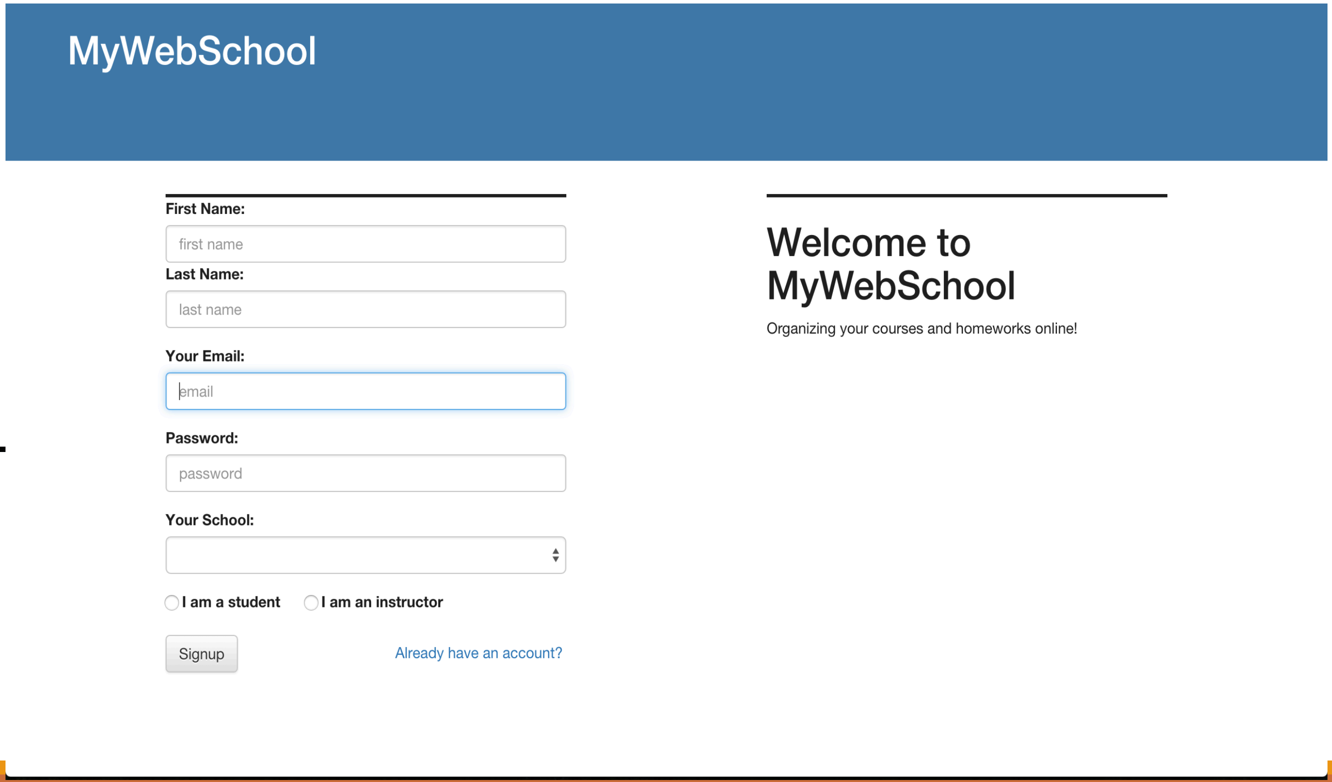
Your Email:
email

Password:
password

Login Forgot your password? Don't have an account?

Welcome to MyWebSchool

Organizing your courses and homeworks online!


MyWebSchool

First Name:
first name

Last Name:
last name

Your Email:
email

Password:
password

Your School:

I am a student I am an instructor

Signup Already have an account?

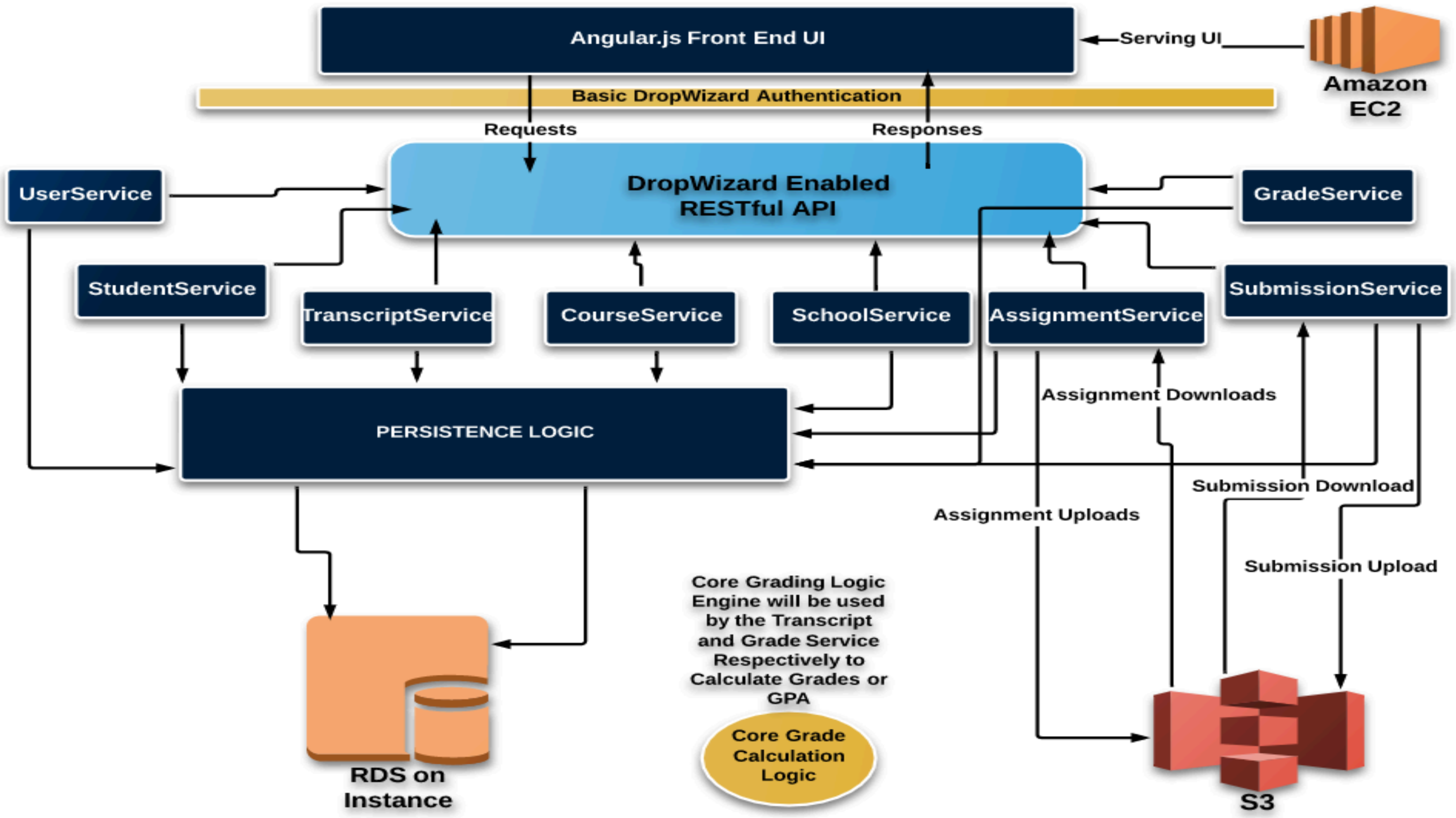
Welcome to MyWebSchool

Organizing your courses and homeworks online!

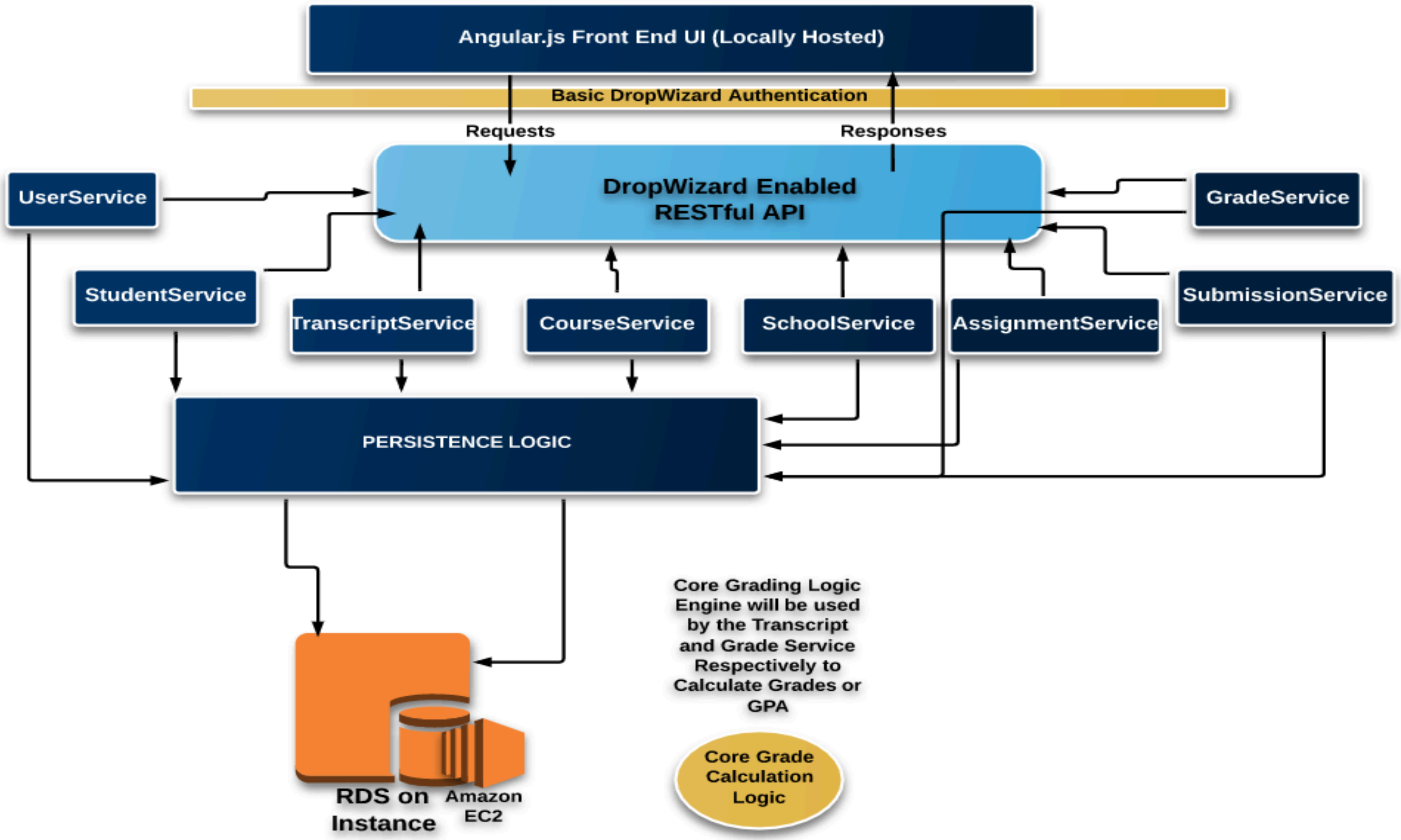
Architecture

CHRISTOPHER LICATA

Envisioned Architecture



Actual Architecture



Angular.js Front End UI (Locally Hosted)
Basic DropWizard Authentication
Requests
Responses
DropWizard Enabled RESTful API
UserService
StudentService
TranscriptService
CourseService
SchoolService
AssignmentService
SubmissionService
GradeService
PERSISTENCE LOGIC
RDS on Instance
Amazon EC2
Core Grading Logic Engine will be used by the Transcript and Grade Service Respectively to Calculate Grades or GPA
Core Grade Calculation Logic

Use of Modular Architecture

- **Every component of the software reflected this**
  - i.e. the core business logic, persistence layer, front facing restful API, and the services
- **Completely Distinct Components**
  - Each provides an interface through which they communicate
  - Core business logic and persistence layer are in different projects, that are then imported to the restful API's project
- **Benefits:**
  - Easy to switch the database from relational to document-based
  - Restful API can communicate with persistence layer via web services
  - Scope of interaction between UI and API is controlled by the restful API

Service Oriented Architecture

- **9 total services**
  - Together act as a facade to persistence layer logic and databse
  - Provides Dropwizard Resources in RESTful API with ability to retrieve, save, and manipulate objects coming in from client side
- **Dropwizard Resources**
  - Constrains scope of the client side UI
    - Due to specification, enforcement objects, and request templates
  - Also serves the POJOs as JSON objects to the client and vice versa

Technical Choices: MySQL (JDBC) over Hibernate

- **Benefits of MySQL**
  - Full control of database and queries
  - Industry standard
    - Widely compatible with most operating systems and easily accessible support
  - Writing large amounts of boilerplate code is not necessary
  - On a large scale (i.e. n > 100,000 rows), there is less of a speed overhead than in comparable programs
- **Drawbacks of MySQL**
  - Less out-of-the-box functionality than other comparable platforms
    - Some features are dependent on additional applications and are not on the core engine
  - Stability Issues in handling functions like references, transactions, and auditing
  - Relatively Poor Performance Scaling
    - Best used when you have low read/write ratio

Technical Choices: MySQL (JDBC) over Hibernate

- Benefits of Hibernate –
  - Reduced Impedance Gap (i.e. Impedance mismatch), which is when the object in your application do not perfectly map 1-1 with your database tables.
  - Additionally, Hibernate affords you the ability of easy cascading.
- Disadvantages of Hibernate –
  - Extremely large learning curve
  - Too abstract – everything happens "*automagically*", which frustrates developers who like to have full control over what occurs.

Technical Choices: Amazon RDS

- **Decisions**
  - AWS RDS with EC2 vs. EC2 with DB and WebServer inside vs. 2 ECS instances (one for DB, other for application hosting)
- **Scale-Up Options when single EC2 server is overloaded**
  - Scale Vertically
    - Upgrade to the next largest instance, by adding more physical resources via control panel
  - Split Server into two EC2 instances
    - Most complicated option to implement
    - Little to no support in terms of configuration, management, monitoring, and patching
  - Migrate Database to Amazon RDS
    - Includes automatic backups and maintenance out of the box
    - Features intangible 15% performance boost over an EC2 MySQL instance
    - Simplest to implement, simply add more physical resources via control panel
- **Final Decisions**
  - Unable to use EC-2 Web server due to time constraints and the high learning curve for the frontend team with this technology
  - This was the original intention for the architecture

Trade Offs: Mockito

- **Benefits:**
  - Allows method call chaining, to produce less imperative looking code
  - Can mock out interfaces and classes
  - Method chaining style interface is easy to write
  - Clean API
  - Little time needed to start mocking process; only one mock type and one method to create mocks
- **Drawbacks:**
  - Potential for difficulties in maintenance
    - Method chaining is easy to write, but difficult to read
    - This can make it difficult to determine why a test, other than the current one being used, has failed
  - Impossible to mock final classes or static classes

Trade Offs: DropWizard

- **Benefits**
  - Can easily write Micro services
  - Automatically gives health stats per service
  - Configuration is in YML, while Spring Config is in XML/Annotations
  - Tens of integrations with other community tools.

- **Disadvantages**
  - Comparable to Spring, the documentation isn't as thorough
  - Authentication was very difficult to understand both at the backend and frontend.
  - Unclear integration with other libraries for JMS, Scheduling, etc.

Trade Offs: JUnit4

- **Benefits**
  - Suite Test Feature: can bundle a few unit tests and run them together
  - Can utilize parameterized tests
    - Allows for variable parameter test for unit test
  - Rather than suite() methods to build test suite out of multiple classes
  - @Before methods automatically called in the right order throughout whole class hierarchy
  - Can temporarily disable tests with @Ignore
- **Downsides:**
  - Limitations to parameterized tests
  - No support for dependency test
  - No distinction made between failures and errors (hard to track down source of problem)

Trade Offs: AngularJS

- **Benefits**
  - Two way data-binding
  - MVC
    - Model/ViewModel
    - Controller/View
  - Dependency injection
    - Sharing data between controllers
  - Directives
- **Downsides:**
  - The documentation is not designed for development
  - The start time increases as the project grows major computational overhead)
  - Complicated to tell which way is better for particular task

Trade Offs: Bootstrap

- **Benefits**
  - Speed of Development
  - Responsiveness
  - Consistency
  - Customizable
- **Downsides:**
  - Massive library
  - Very recognizable
  - Fairly difficult to customize

Easily Changeable Parts

- **Completely Distinct Parts**
  - Including the core business logic, persistence layer, front facing restful API, and the service facade to and from the RESTful API
  - Provides an interface through which they communicate
  - Each component focuses on necessary operations for one specific data access object
  - Core business logic and persistence layer stored in different projects to import into the restful API's project
- **Benefits**
  - Allows the database to switch from relational to document based
  - Restful API communicates with persistence layer via web services
  - Allows for easy scaling, manipulation, and removal of software components

Development

Easiest Aspects

- Designing the database schema
- Deploying an RDS instance
- Developing the queries for the CRUD operations for all DAO objects
- Use of relational databases, over a document-based one
  - Lent itself more readily to tabular data

Challenges in Development

- Discrepancy in caliber of expected work
  - Professional Developers vs. Nonprofessional Developers
- Time Constraints
  - Accommodating schedules of 4 full-time students, one of which works full-time
- Novel Concepts and Applications
  - Side Effect free functions
  - Java 8 and Mockito
  - Some test-driven development
  - Dropwizard and authentication
  - Angular ui-route
- Unexpected difficulties rooted in database selectin
  - Constantly changing models increased the development time of the persistence

Decisions in Hindsight

- Implementation of sprint schedules and deliverable dates for the team
- Creation of a realistic timeline
- Overall fundamental misunderstanding of LOE's, realistic timeline, and the necessity of daily communication

Hindsight is always 20/20

Work Distribution

- **Plan**
  - Two members on the front end, two members on the backend
  - Would have allowed flexibility in task distribution and time management
- **Reality**
  - Low attendance of team members to class resulted in fewer team meetings and contributions
  - External conflicts led to some members not participating at all
  - Attempted to resolve these issues once they became apparent, but was unsuccessful
  - Ultimately the work was redistributed: two members on the front end and one member on the backend

Challenges Working as a Group

- Varying degrees of participation among team members
- Apathy toward group determined time lines and work distribution
- Overall result of this was a reallocation of tasks to complete this

Future Steps

- **Backend**
  - Refractor the code base
  - Complete the rest API
  - Add caching and transactional log support
- **Front End**
  - Utilize unit-testing for the angular.JS code
  - jQuery and Bootstrap to continue enhancing the user’s experience