**InstructED**

**Software Design**

**CSCI-P465/565 (Software Engineering I)**

**Project Team**

**Kevin Cao**

**Andrew Jedlicka**

**Blake Miller**

**Shaun Trimm**

**1. Introduction**

* 1. **System Description**

Our project is a Learning Management System implemented as a web app. We are aiming to create an application that allows instructors and students to interface, allowing for completion of assignments and grading of work.

**1.2 Design Evolution**

**1.2.1 Design Issues**

We want our app to run on any operating system using web browsers that support React and Node.js. Our app requires access to the internet, which will be a requirement. Our app is also intended only for use on personal computers rather than on cell phones or other interfaces, but hopefully will be usable on some other devices.

**1.2.2 Candidate Design Solutions**

Briefly (a paragraph or two) describe each of the possible design solutions that your team discussed/explored.

Our first decision was on our front-end: we debated between using HTML + CSS versus Javascript. We briefly discussed using other front-end architectures. We debated between React, Angular, and Vue. However, we did not discuss functionality on other systems, such as phones or tablets. We did discuss use of other database systems, notably comparing PostgreSQL and MySQL. These solutions would have likely resulted in a similar outcome to what we chose to use for the final version of our project.

**1.2.3 Design Solution Rationale**

We chose JS because we believed it to be the most powerful and well-documented solution. We used similar logic to choose React, which we chose above Angular and Vue because it was more well-documented and established. PostgreSQL was chosen over MySQL because Heroku, the hosting solution we chose, had support for it rather than MySQL.

**1.3 Design Approach**

**1.3.1 Methods**

We have used design documents as a primary way to maintain our design vision. This has allowed us to modify our design while maintaining a consistent “bible” of design choices we have made, while also giving us an easy way to demonstrate design needs to our customer.

**1.3.2 Standards**

We have consistently used camel-casing for our naming convention. Additionally, we have used a standard page format with a navbar that is consistent across all pages. Finally, our tables adhere to Third Normal Form for consistency.

**1.3.3 Tools**

We are using Heroku to host our site, which does not generate anything other than a working copy of our git branch on a remote site.

**2. System Architecture**

**2.1 System Design**

Our site allows for our three types of users, which include students, instructors, and administrators to access a variety of learning tools. Students can create accounts, message professors, complete assignments, view their schedule,

Students can do the following:

* Reset their password, modifying their user info by using a reset-password link sent via email
* View all their courses as well as the assignments in their courses that are upcoming
* Complete assignments, creating an entry in the table for that assignment for their submission which contains student id, assignment id, submission, submission date, and grade (which is initialized to null)
* Chat with other students and professors
* Search for assignments
* Change password

Instructors can do the following:

* View all their courses
* Create assignments, creating a row in a table that contains the course id, assignment id, due date, description of assignment, and submission type
* View individual submissions for assignments and assign them a grade, modifying the entry in submissions table for a given student’s submission
* Chat with students
* Change password

Admins can do the following:

* Change role of existing user
* Enroll student in a course
* Create a course and assign an Instructor to it

**2.2 External Interfaces**

We are currently using Duo authentication’s interface. We have been using this interface to secure our logins, accessing Duo only when we have a successful username/password entry pair. We are also using Facebook authentication, which allows for users to login using Facebook.

**3. Component Design**

**Component Name**  
Login and Registration

* **Component Description**  
  The main page on our site prior to login is the login prompt. The login prompt allows for users to log in using existing credentials, create a new account, login with Facebook, or reset their password. If a user logs in, they are redirected to Duo, which redirects if successful to the dashboard. If they want to create an account, they are directed to that page where they input account information. If they login with Facebook, it is similar to regular log-in. If they choose to reset their password, they enter their email, then go to the emailed URL where they enter their username and new password twice to reset it.
* **Responsible Development Team Member**  
  Kevin Cao
* **Component User Interface**  
  Users will receive a message if they incorrectly log in, enter non-matching passwords during registration or password resetting, or fail to login with Facebook or Duo. User screens are login, register, forgot password, and reset password.
* **Component Objects**
* DuoLogin
  + Passes DuoAuthState to login class, checking if duo auth succeeds
* ForgotPassword
  + Posts to /forgotPassword
* Login
  + Posts to /authenticate, checking db for password
  + Posts to /authenticate/facebook, checking for successful facebook validation
  + Pushes user to /duologin if successfully authenticated
* Navbar
  + Accesses Login, Create Account, and Forgot Password
* Registration
  + Posts to /users
  + Posts to /authenticate/facebook
* ResetPassword
  + Posts to /updatePassword
  + Gets from /resetPassword/(token)
* Queries:
  + CreateUser updates Users with insert
  + CreateLoginToken creates login token from userID
  + LoginUser checks Users table using email and password
  + loginFacebook checks Users table using email
  + loginFacebook inserts into users email and name
  + ForgotPassword checks Users
  + ForgotPassword creates password reset token
  + CreatePasswordRestToken inserts into PasswordTokens
  + CreatePasswordResetToken updates PasswordTokens
  + CheckResetExpiration checks PasswordTokens
  + UpdatePassword checks PasswordTokens
  + UpdatePassword updates Users
* **Component Interfaces (internal and external)**  
  This component will provide all other components with the current user. This will include their user ID, name, type, and anything else. This component is the first to be loaded, and all other components require this to function. This also interfaces with our Duo and Facebook implementations.
* **Component Error Handling**  
  Error Case 1: Input for username can be SQL injection, prevented by ensuring entry is properly formatted and inserted rather than added to query  
  Error Case 2: Users can access URL’s without being directed to them by our application so we look for queries and cookies to ensure no user can get into another account  
  Error Case 3: Users can input incorrect passwords or inconsistent passwords so we notify them if this is the case and do not allow them to access sensitive information
* **Component Name**  
  Dashboard
* **Component Description**  
  After logging in, the user is presented with the Dashboard view. This view allows for the user to set “widgets” which contain specialized content. Users can choose to have a calendar view, announcements, assignments, and more as their widgets. These display information posted from classes. Additionally, different users have different permissions; the admins have access to create classes and manually add students to courses, instructors have access to creating assignments and announcements, and more.
* **Responsible Development Team Member**  
  Andrew Jedlicka.
* **Component User Interface**  
  Users can navigate to individual course pages from the sidebar. Otherwise, they have just access to the four widgets on the main page.
* **Component Objects**  
  Describe the objects/classes that comprise this component.  Provide a listing of expected data members and methods for each class.  Note in the description if a given object/method does any of the following (Note: Some of this information may be captured in class diagrams or other parts of the design - simply be sure it is included somewhere):
  + Makes an operating system call (cite expected system calls to be made)
  + Makes a hardware-specific system call (cite calls to be made)
  + Creates/alters/deletes a file (cite file names)
  + Explicitly calls the method of another object (cite name of other object)
  + Is explicitly called by another object (cite name of other object)
  + Passes data to another object (cite data structure and name of other object)
  + Receives data from another object (cite data structure and name of other object)
  + Is derived from another object (cite name of other object)
* AddCourse
  + Posts to /courses
* Announcements
  + Gets from /announcements to get all announcements for all courses (admin only)
  + Gets from /announcements/instructor for instructors to view all announcements from all their courses
  + Gets from /announcements/instructor and announcements/student to view all announcements from all the student’s courses and all courses that they are a student in
* AssignmentsPerDay
  + Gets all assignments on a given day from Assignments
* CourseAndInstrList
  + Gets all courses, their instructors, and their term from /courses
* CreateAnnouncement
  + Gets from /roles to see role of user
  + If admin, gets all courses from /courses
  + If instructor or student, gets all courses they instruct from /courses/instructor
  + Posts to /announcements using values from input
* TaskCalendar
  + Gets from assignments/date using a date
  + Shows users which assignments are due on a given date
* UpcomingAssignments
  + Gets from /roles
  + If admin, gets all assignments from /assignments
  + If instructor, gets all assignments from /assignments/instructor for courses instructing
  + Else, gets all assignments for student
  + Gets deadlines as well as assignments
* Queries:
  + GetRole gets from Roles using userId
  + GetAllCourses:
    - Gets from Courses if admin
    - If Student, gets from Enrollments + Courses inner join on course ID
    - If instructor, gets from Instructing + Courses inner join on course ID
  + GetAllAnnouncements:
    - Gets from Announcements if admin
    - If Student, gets from Announcements join with (Courses inner join Enrollments on courseID) on courseID
    - If Instructor, gets from Announcements join with (Courses inner join Instructing on courseID) on courseID
  + GetAllAssignments:
    - Gets from Assignments if admin
    - If Student, gets from Assignments join with (Courses inner join Enrollments on courseID) on courseID
    - If Instructor, gets from Assignments join with (Courses inner join Instructing on courseID) on courseID
  + GetAssignment gets an assignment from Assignment table given an assignment ID
  + userCanAccessAssignment gets from assignments inner join courses on course\_id and returns bool if student/instructor is enrolled in or instructing course
  + queryForRole returns a user’s role
  + GetCourseAnnouncements gets from Announcements table using course ID
  + GetCourseAssignments gets from Assignments table using course ID
  + AddAnnouncement posts to Announcements table using name, description, course ID, date, and userID
  + GetAssignmentsByDate gets from Assignments table
* **Component Interfaces (internal and external)**

This component will allow navigation to other components, including assignment upload, search, and chat. This component will load immediately after login, and can be navigated to from any other page.

* **Component Error Handling**  
  Error Case 1: Input for several tables can be SQL injection, prevented by ensuring entry is properly formatted and inserted rather than added to query  
  Error Case 2: Users can access URL’s without being directed to them by our application so we look for queries and cookies to ensure no user can get into another account  
  Error Case 3: Users can input long text values for things like assignment names so we limited size of these fields
* **Component Name**  
  Submission and Grading
* **Component Description**
  + This section of our project pertains to the creation of assignments, creation of submissions, and grading. For this component, our users can use widgets that allow for all of this functionality. This component primarily focuses on the user experience for Students and Instructors.
* **Responsible Development Team Member**  
  Andrew Jedlicka
* **Component User Interface**  
  With this component, we have continued the Widget-based design outline that we created in the previous sprint. We have widgets for each of our get requests. This allows for full use of everything in this sprint.
* **Component Objects**  
  Describe the objects/classes that comprise this component.  Provide a listing of expected data members and methods for each class.  Note in the description if a given object/method does any of the following (Note: Some of this information may be captured in class diagrams or other parts of the design - simply be sure it is included somewhere):
  + Makes an operating system call (cite expected system calls to be made)
  + Makes a hardware-specific system call (cite calls to be made)
  + Creates/alters/deletes a file (cite file names)
  + Explicitly calls the method of another object (cite name of other object)
  + Is explicitly called by another object (cite name of other object)
  + Passes data to another object (cite data structure and name of other object)
  + Receives data from another object (cite data structure and name of other object)
  + Is derived from another object (cite name of other object)
* GetRole
  + Gets from Users
* SetRole
  + Sets the role of a user using a given UserId
* GetAllCourses
  + For instructor or student, get all classes enrolled/instructing
  + For admin, get all classes in the Courses table
* GetAllAnnouncements
  + For instructor or student, get all announcements for classes enrolled/instructing
  + For admin, get all Announcements from all courses
* GetAllAssignments
  + For instructor or student, get all assignments for classes enrolled/instructing
  + For admin, get all assignments from all courses
* GetCourseAssignments
  + Gets from /assignments using a courseID
* GetCourseAnnouncements
  + Gets from /announcements using a courseID
* GetAssignmentsByDate
  + Gets from /assignments using an input date
  + For instructor and student, only gets assignments for current classes
  + For admin, gets for all classes
* GetUpcomingAssignments
  + Gets from /assignments using the current date
  + For instructor and student, only gets assignments for current classes
* GetAssignment
  + Gets from /assignments given assignmentID
* GetCourse
  + Gets from /courses given CourseID
* addCourse
  + Puts to /courses using course dept, number, name, and term
* addInstructorToCourse
  + Puts to Instructing using userId and courseId
* addStudentToCourse,
  + Inserts into Enrollments using userID and courseID
* addAnnouncement,
  + Inserts into /announcements using given name, description, courseID, userID, and current date
* addSubmission,
  + Inserts into /submissions using assignmentID, userID, and current time
* addLinkSubmission,
  + Inserts into /submissions using assignmentID, userID, current time, and a link
* addAssignmentFile
  + Inserts into /submissions using assignmentID, userID, current time, and files
* addCourseVideo,
  + Inserts into CourseFiles using courseID, name, and current time
* getCourseVideos,
  + Gets from CourseFiles using courseID
* addCourseFile,
  + Inserts into CourseFiles using courseID and name
* getCourseFiles,
  + Gets from CourseFiles using courseID
* getCourseStudents,
  + Gets from Enrollments inner join on Users using a given CourseID
* getAssignmentSubmissions,
  + Gets from submissions using userID and assignmentID where there is no link
* getAssignmentFiles,
  + Gets from AssignmentFiles using assignmentID
* getGrade,
  + Gets from Grades given userID and assignmentID
* addGrade
  + Puts into Grades given userID and assignmentID
* addCourseVideo
  + Puts into CourseVideos given filename, path, courseID, and current time
* getApprovedCourseVideos
  + Gets from CourseVideos where approved is true
* getCourseVideos
  + Gets from CourseVideos either using a courseID or using no input
* getUnapprovedCourseVideos
  + Gets from CourseVideos where approved is false
* addAssignmentFile
  + Puts into AssignmentFiles using a name, path, assignmentID, and filename
* getApprovedAssignmentFiles
  + Gets from AssignmentFiles where they have been approved
* getAssignmentFiles
  + Gets from AssignmentFiles using an assignmentID
* getUnapprovedAssignmentFiles
  + Getts from AssignmentFiles where they have not been approved
* getApprovedCourseFiles
  + Gets from CourseFiles given a courseID or nothing
* getUnapprovedCourseFiles
  + Gets from CourseFiles using a courseID where not approved
* getRoleInCourse
  + Gets from Instructing using a userID and courseID
* approveCourseFile
  + Updates value in CourseFiles using a given courseID and filename
* approveCourseVideo
  + Updates value in CourseVideos using a given courseID and filename
* approveAssignmentFile
  + Updates value in AssignmentFiles using assignmentID and filename
* setCourseFileApproval
  + Updates value in CourseFiles using courseID and filename
  + Approves file or deletes file
* setCourseVideoApproval
  + Updates value in CourseVideos using courseID and filename
  + Approves file or deletes file
* setAssignmentFileApproval
  + Updates value in AssignmentFiles using assignmentID and filename
  + Approves file or deletes file
* getSyllabus
  + Gets from syllabuses using courseID
* updateSyllabus
  + Updates value in syllabuses using syllabus input and courseID
* **Component Interfaces (internal and external)**  
  This component can affect the search and chat features as it can modify instructors, enrollments, assignments, and files. Many of these can be searched through, or otherwise affect chat by increasing the pool of users in a course.
* **Component Error Handling**  
  Error Case 1: Input for many of our tables can be SQL injection, prevented by ensuring entry is properly formatted and inserted rather than added to query  
  Error Case 2: Users can access URL’s, including queries, without being directed to them by our application so we look for queries and cookies to ensure no user can get into another account  
  Error Case 3: Users can input invalid files so we prevent these from being added to our DB
* **Component Name**  
  Search
* **Component Description**  
  This feature extends to most other components and allows for users to query the site to find specific content using keywords. Users can access this currently from users and courses, and will be able to also query files, assignments, and submissions as well. Every type of user can use this feature. When inputting into the search bar, the search bar will also attempt to auto-complete, finishing the query with a guess about its content based on the content of the database.
* **Responsible Development Team Member**  
  Shaun Trimm
* **Component User Interface**  
  If this component includes a user interface, include some details about the interface including what aspects of the component will be available through the interface, a description of each of the user screens that are expected for this component, and a description of each of the user notifications and/or messages that are planned for this component.  
  This component consists of a search bar that is implemented within several pages of our document. They can click within the search bar to begin typing a query for our page, which will automatically, upon typing, query our database to find relevant results. Users can also choose filters to see only results of certain types.
* **Component Objects**  
  Describe the objects/classes that comprise this component.  Provide a listing of expected data members and methods for each class.  Note in the description if a given object/method does any of the following (Note: Some of this information may be captured in class diagrams or other parts of the design - simply be sure it is included somewhere):
  + Makes an operating system call (cite expected system calls to be made)
  + Makes a hardware-specific system call (cite calls to be made)
  + Creates/alters/deletes a file (cite file names)
  + Explicitly calls the method of another object (cite name of other object)
  + Is explicitly called by another object (cite name of other object)
  + Passes data to another object (cite data structure and name of other object)
  + Receives data from another object (cite data structure and name of other object)
  + Is derived from another object (cite name of other object)
* searchUsers
  + Selects from Users
  + Parameters include a first + last name and a user-role, represented as an integer
* searchCourses
  + Selects from Courses
  + Parameter is a course name
* **Component Interfaces (internal and external)**  
  This component will interface with all relevant searchable areas of our program, including users, courses, and assignments.
* **Component Error Handling**  
  Error Case 1: Search query can have no results, so we log an error if that is the case and return no results
* **Component Name**  
  Chat
* **Component Description**  
  This feature allows for individual users to chat with other users or groups. Users can create new chats as well as continue chats that have been ongoing. Users can see when other users are online as well as when they are typing. They can also see messages in real-time.
* **Responsible Development Team Member**  
  Kevin Cao
* **Component User Interface**  
  Users can navigate to chat from a course page. They can also navigate back to the course page from chat using an X that is at the top-left or by clicking “back to courses.” They can then select chats from a sidebar on the left. Users can see chat messages that they have sent on the right side of the screen, accompanied by a timestamp, and messages from other users on the right with a timestamp and name.
* **Component Objects**
* getUserConversations
  + selects from UserConversations using conversation ID and course ID
  + gets users from said userConversations
* getConversationMessages
  + gets from Messages and Users based on a conversation ID
  + gets the content of messages
* addMessage
  + Inserts into Messages using a conversation ID, message, sender ID, and send time
  + Sends a message into a conversation
* createConversation
  + Inserts into conversations and UserConversations
  + Takes conversation ID and recipient ID
* searchCourses
  + Selects from Courses
  + Parameter is a course name
* checkOnlineStatus
  + gets from OnlineUsers using user\_id
* **Component Interfaces (internal and external)**  
  This component interfaces with the course page.

**Revision History**

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Change Description** |
| 1 | 10/4/20 | Begun and added Login/Registration |
| 2 | 10/18/20 | Added Dashboard |
| 3 | 11/01/20 | Added Submissions and grading |
| 4 | 11/15/20 | Added Search as well as updates to submissions and grading |
| 5 | 11/16/20 | Modification to Search based on implementation |
| 6 | 12/06/20 | Added Chat and modified Search based on new input |
|  |  |  |
|  |  |  |
|  |  |  |

**Page Author/Creator:** Blake Miller

**Last Modified:** 12/06/20