Design Document

Theo Burkhart
Todd Griffin
Justin Hartman
Tyler Rex
Mason Wesolek

Team 3

Purpose

Currently, the majority of classes on Purdue's campus use iClickers to administer quizzes electronically in class. Despite its popularity, the iClicker system has major drawbacks. iClickers only allow students to respond to quizzes with 5 different choices, limiting the type of quizzes to multiple choice. Using iClickers in class also requires students and schools to purchase proprietary hardware. Alternatives such as Kahoot and HotSeat are available which allow students to respond to quizzes with any internet connected device. However, Kahoot has limited functionality given it's marketed towards young kids, and HotSeat allows text input but it's limited to 100 characters.

Our project aims to bridge the gap between these products and design a fully-featured in class quiz system that satisfies both instructors and students. Quack will allow students to take quizzes and participate in class through an iOS app, Android app, or a web browser. Both the app and web app will enable students to take multiple choice, extended response, fill-in-the-blank, or matching quizzes as well as respond to live polls. Students will be able to view past quiz results and scores from all their classes. Instructors will have a powerful web app to create quizzes, make classes with multiple sections, register students, and allow TAs to access and grade quizzes. Our hope is that Quack will be a fully-featured and robust system for administering quizzes in class.

Design Outline

Our system will use a client-server model. Clients will be made up of Android users, iOS users, and web app users. Using React and React Native we will be able to reuse much more code across different platforms than if we developed separate native apps. The server will host our API and handle requests between the database and the client. Our server, built with NodeJS/Express, will service API requests using GraphQL. Microsoft Azure will handle our SQL database.

1. Client
    a. The client will send an API request to the server
    b. Client receives a JSON response from the server
    c. Our client will then render needed information using React/React Native
2. Server
    a. Receives API requests from client and validates the client
    b. Queries database
    c. Sends client back requested data
3. Database
    a. Receives queries from the server
    b. Manipulates or retrieves data as necessary
    c. Responds back to server

Design Issues

Functional Issues:

1. **How is the student found to be physically present?**

   Option 1: Use GPS location to check if physically in the classroom
   Option 2: Show a code on the screen that students enter before taking quiz
   Option 3: Use Bluetooth to check in to the class
   Option 4: Have TA/professor manually check in

   Decision: We chose to primarily use a GPS location to see if the student is present in the classroom, and then as a backup show a code that recycles approximately every 10 seconds on the screen if the location cannot be found. This was chosen so we can provide the fastest and most seamless experience possible when it is time for a quiz. Letting the students and professor be as efficient as possible while cutting down on academic dishonesty.

2. **How does a student register for a class?**

   Option 1: Professor will have a unique link that students can use to register
   Option 2: Manually assign students to the class
   Option 3: Students can search for courses and sections and enroll

   Decision: We decided that a professor will have a unique link that students can use to register for a class. This can be convenient for professors and students due to the pre-semester emails that are usually sent out and the piazza links that are also used. When clicking on the link it will either bring them to the website to confirm or the mobile app itself to confirm.

3. **What are the roles/permissions?**

   Option 1: Professor will be able to assign roles and permissions
   Option 2: Give roles and permission defaults

   Decision: Giving the professor the ability to assign all the roles and permissions allows the class to work the same way that it has always been structured. This

allows a seamless transition from the way that the class used iClickers or other ways of quizzes and managing to Quack.

Non-Functional Issues:

1. **What backend framework should we use?**

   Option 1: NodeJS/Express
   Option 2: Java
   Option 3: Python

   Decision: While we have experience with all three backend technologies, we felt that NodeJS/Express would best fit our frontend needs and fit better with the overall system that we are using. Since we are using GraphQL and other modern technologies, we figured it would be easiest to use the framework that is most commonly provided with setup guides and is quickly becoming one of the industry standards for web development.

2. **What frontend framework should we use?**

   Option 1: React/React Native
   Option 2: Seperate native apps (Swift/Java)
   Option 3: Xamarin
   Option 4: Angular

   Decision: We decided to use React/React Native for our whole suite of apps. With the Javascript backend there is a plethora of resources that can be used throughout the app, and cross platform codebase allows us to write code once for both platforms.

3. **What database provider should we use?**

   Option 1: Microsoft Azure
   Option 2: AWS RDS
   Option 3: MongoDB

Decision: While we have experience with MongoDB, we decided that an SQL database would fit our project's needs better than a noSQL database. This left us with the options of AWS RDS and Microsoft Azure. Since we get free benefits with Azure as Purdue students, we believe it will be more cost-effective to use Microsoft Azure.

4. **What type of architecture should we use?**

   Option 1: Client/server architecture
   Option 2: Unified architecture
   Option 3: Peer-to-peer architecture

   Decision: Client/server architecture we felt would be the best way to structure the development process and the best for the app itself. Relying on the server to manage quizzes and permissions allows the front-end and back-end independently and then come together to make sure everything works together.

Design Details

Database Diagram

Description of Database:

Above is a diagram of our database functionality. While things can change throughout the semester, this will be our guideline as we progress.

1. User
    a. Each user is represented similarly within the database, whether they are an instructor, TA, or a student.
    b. Each user will have a name and email attached to their account.
2. Course
    a. Each course will have an ID and a name associated with the course.
    b. This will be used to group sections, quizzes, and keep items organized on the user end.
    c. Courses are created by instructors.
3. Role
    a. Roles will keep track of permissions that a user has for a specific course.
    b. Roles will be custom-made by the owner of the course (instructor).
    c. A single user can have multiple roles for different classes.
4. Section
    a. Each section will only have one course.
    b. Sections will be used to further maintain the structure that is implemented by many classrooms that have more than one section per course.
5. Quiz
    a. Quizzes will be created by instructors or users that have permission to (TAs).
    b. Each quiz is associated with a section and will data that stores if it is currently open, the date it was presented, and the correct answer.
    c. Quizzes will also have many types that can be associated with it and will also be used for taking attendance.
6. Answer
    a. Each answer has a type
    b. An answer is tied to one user and one quiz

Navigation Flow Map:

Our navigation map shows that our app will be simplistic to maneuver through. With one main screen being the centerpiece of the app, it will be easy to navigate through each app with only a click. Whether the user is a student or an instructor, the app will dynamically change its screen content to fit the needs of the user. For example, when a student has a quiz that needs to be taken, a quiz notification will pop up on the main screen, which is shown in our UI mockups.

When a user opens the app:

When a user takes a quiz:

When an instructor creates a quiz:

GraphQL Schema:

Since we are using GraphQL to query our database, we will only need a single URL endpoint: /graphql. Unlike REST, we have no need for multiple endpoints to handle all requests. Instead, we created a GraphQL Schema that will be used to handle requests from our single endpoint.

```
type User {
        id : ID!
        name : Name
        email : String!
        courses : [Course]
        roles : [Role]
}

type Course {
        id : ID!
        title : String!
        instructors : [User]!
        sections : [Section]!
        Quizbook : [Quiz]
}

type Section {
        id : ID!
        title : String!
        course : Course!
        roster : [User]!
        quizzes : [Quiz]
}

type Quiz {
        id : ID!
        prompt : String!
        answers : [Answers]
        accesscode : String!
}
```

```
type Answer {
        id : ID!
        user : User!
        content : [String]!
}

type Role {
        id : ID!
        section : Section!
        title : String!
        permission : [Permission]
}

type Permission {
        content : String!
}

type ID {
        id : String!
}

type Name {
        first : String!
        middle : String
        last : String!
        full : String!
}
```

UI Mockups

Student Mobile App UI

Student Web App UI

Instructor Web App UI