Description

Quizzly is a platform that provides convenience for professor to give out a quiz and for students to take it in class using their smartphones.

Stakeholder:
Reem Alfayez

Team:
Rahul Chander
Eddie Hays
Jacob Rey
Anthony Tu
Wilson Yeh

Installation

Back-end

A Sails application

A. Install NodeJS: NodeJS installation

B. Clone quizzly-backend (Git Commands Guide w/ Clone Command)

C. cd into directory quizzly-backend

D. sudo npm install (Installs dependencies using npm package manager)

E. Set up an empty MySQL Database somewhere (locally, Amazon RDS, on the server, etc.)

F. Create a .env file (https://www.npmjs.com/package/dotenv) OR directly configure environment variables

Environment Variables Needed:

- MYSQL_HOST= [MySQL DB Host URI]
- MYSQL_USERNAME= [MySQL DB Username]
- MYSQL_PASSWORD= [MySQL DB Password]
- MYSQL_DATABASE= [MySQL DB Database Name]
- NODE_ENV=production
- MONGODB_URI= [MongoDB URI if using MongoDB, make sure to edit database.json!!!]
- APN_CERT= [APN Certificate for Apple Developer Push Notifications
- APN_KEY= [APN Key for Apple Developer Push Notifications]

G. Initialize the database

1. Open ‘quizzly-backend/config/models.js’
2. Change migrate: ‘safe’ to migrate: ‘drop’ (Enables migrations, wipes database)
3. sails lift (Starts the server, also initializes the database based on sails models)
4. Change migrate: ‘drop’ back to migrate: ‘safe’ (Disables migrations)
5. db-migrate reset (Clear any *db-migrate* migrations)
6. db-migrate up (Runs migrations to seed the ‘term’ and ‘year’ tables, https://github.com/db-migrate/node-db-migrate)

H. Make sure your host’s firewall is allowing for HTTP requests on port 1337 so Sails can receive them.

Front-end

A. Install NodeJS: NodeJS installation

B. Clone quizzly-frontend (Git Commands Guide w/ Clone Command)

- Usually, clone to /var/www/html/

C. cd into directory quizzly-frontend

D. sudo npm install (Installs dependencies using npm package manager)

E. Create a .env file (https://www.npmjs.com/package/dotenv) OR directly configure environment variables. Environment variables needed:

- NODE_ENV=production

F. Point your web-server (e.g. apache) to <INSTALL_DIRECTORY>/quizzly-frontend/public/

- Usually, <INSTALL_DIRECTORY> = /var/www/html/

G. Modify the urls at the bottom of quizzly-frontend/src/modules/Api.js to point to where you are hosting quizzly-backend (may be the same hostname/server as quizzly-frontend)

Run

Back-end
A. cd into the root directory of /quizzly-backend/

B. sails lift (Starts the server, listening for requests on port 1337)

- Use ./start_server.sh and ./stop_server.sh to start/stop forever without console output
- https://github.com/foreverjs/forever

C. Now you can make HTTP requests to where quizzly-backend is hosted on port 1337

D. Check routes.js for the API, and check quizzly-frontend/src/modules/Api.js for examples of requests using Sails models (these are not explicitly listed in routes.js)

E. Whenever you make a backend change, you must restart the server

Front-end
A. Visit your web server using a browser

B. Whenever you make a frontend change, cd to the root directory of /quizzly-frontend/ and run

```
sudo ./node_modules/.bin/webpack
```

*This minifies the code into bundle.js to be served by the web server

Contribute

Special thanks to Professor Miller and Reem Alfayez for supporting the project.

License

The project is a USC CSCI 401 Capstone Class Project.

The following documentation servers as a reference to the project structure and function that has been written into the front and back ends of the Quizzly web platform.

1. File Structure
2. Detailed File Explanation

Back-end
1. File Structure

api - Where all the important code is.

- controllers

    - Contains business logic functions, used to process data for the associated model and its related tables, ex: CourseController.js can call Course.find() to find a row in the 'course' table.
        - **AuthController:** Special controller without an associated model, used for user authentication using JWT tokens.
    - Frontend can make requests to a controller's functions by making a POST request to the route 'model/function', ex: Api.db.post('question/answer').
    - Alternatively, you can create custom routes in config/routes.js.
    - Sails docs: http://sailsjs.com/documentation/concepts/controllers
- models
    - Each model is a definition of a table in the database.
    - Sails automatically generates these tables on lift when set to migrate: 'drop'.
    - Frontend can make basic POST requests to select/insert/update/delete rows of a table by using the model's name in the URI.
        - Example: POST to 'QUIZZLY_URL/professor/find/' with {id=1} in the body data returns the row in the 'professor' table with id=1.
        - Implemented already in quizzly-frontend, make calls using **Api.js**.
        - **Caution:** Could be dangerous to ignore table relationships, check routes.js for functions you could call instead.
    - Sails docs: http://sailsjs.com/documentation/concepts/models-and-orm/models
- policies
    - Authorization policies, such as allowing only Students to answer questions.
    - Sails docs: http://sailsjs.org/#!/documentation/concepts/Policies
- responses
    - Customize your HTTP response codes
    - Sails docs: http://sailsjs.com/documentation/concepts/custom-responses
- services
    - Similar to controllers, but model agonistic.
    - Can be used from anywhere, good for DRY practices between controllers.
    - Sails docs: http://sailsjs.com/documentation/concepts/services

assets - Don't worry about it, assets should be served by Apache and handled in React in quizzly-frontend

config - Ask your stakeholder for any needed keys and certificates.
- Configure your urls and routes.js here

deprecated_migrations - Tried to use db-migrate for migrations, but sails auto-generates some tables. Just let Sails handle it.

docs - Deprecated setup instructions from previous years, but could still be useful

migrations - Migration files for db-migrate. Only create db-migrate migrations to seed tables.

tasks - Grunt configs, check the directory's README if you're interested

views - Don't worry about it, views should be served by Apache and handled in React in quizzly-frontend

2. Models and Controllers

Answer.js and AnswerController.js

- Student answers to questions

AuthController.js

- Handles user authentication, has no model

Course.js and CourseController.js

- Courses, e.g. CSCI-401, created by the Professor
- Contains Sections and Quizzes

Device.js and DeviceController.js

- User's mobile device id
- Received on login from iOS app
- Used for iOS push notifications through APN

Lecture.js and LectureController.js

- Lectures created by the Professor
- DEPRECATED

LectureItem.js and LectureItemContainer.js

- Quizzes and Questions associated with a lecture
- DEPRECATED

MailController.js

- Used to send emails
- **UNUSED**

Professor.js and ProfessorController.js

- Professor user and associated information

Question.js and QuestionController.js

- Questions created by a Professor
- Organized into Quizzes
- Professors can ask a Question, Students will be alerted to Answer it.

Quiz.js and QuizController.js

- Quizzes created by a Professor for a Course
- Can contain several Questions
- Named Quiz because of a former feature to ask an entire Quiz's Questions all at once

Season.js and SeasonController.js

- Four rows: Fall, Spring, Winter, Summer
- Used as part of defining a Term
- Seeded using db-migrate, seedYearSeason.js

Section.js and SectionController.js

- Sections created by the Professor as part of a Course
- Contains Quizzes and Students

StudentAnswer.js and StudentAnswerController.js

- Logs student answers
- id of answer corresponds to Answer

Student.js and StudentController.js

- Student user and associated information
- Can be added to a Section by a Professor

Term.js and TermController.js

- Terms like Spring 2017
- Consist of a Season and a Year

Year.js and YearController.js

- 2016, 2017, etc.
- Used in Term

Front-end

1. File Structure

2. Detailed File Explanation

AddCourseBody.js

Serves as the container to add a course when logged in as a professor. The professor imports the appropriate information and the course will be added to their list of courses.

AddQuestionBody.js

Serves as the container to add a question to a quiz when logged in as a professor. The professor can select a multi-choice or free response question along with a time limit and the correct answer.

AddQuizBody.js

Serves as the container to add a quiz to a section/course when logged in as a professor. The professor has the ability to name the quiz

AddStudentsBody.js

Serves as the container for adding students to a section when logged in as a professor. The professor has the ability to add students by email. The user must be registered in order for them to be added to the section.

AnswerQuestion.js

Allows the student the answer the question as either multi-choice or free response. This file them updates the student's statistics with their profile on the mobile app and the professor's profile.

AnswerQuiz.js

Allows the student to answer an entire quiz by the teacher. This feature is no longer supported with the incorporation of the PowerPoint plugin.

AskStudentQuesiton.js

Allows the student to answer a single question from the professor when it is asked either thru the web or thru the PowerPoint plugin.

Course.js

Serves as the course object for each student and professor. Each student is assigned to a course by a professor.

Courses.js

Serves as a mechanism to bring all the courses together. Each student has a respective list of courses and so does the professor.

Download.js

Allows the user to download the standalone application that could be used by the professor to ask students questions/quiz. This feature however is no longer supported because of the introduction of the PowerPoint plugin.

DonutComponent.js

Dummy data.

EditCourseModal.js

Allows the professor to edit a course that he/she has already made. The update is then reflected on the server as well.

EditSectionModal.js

Allows the professor to edit a section for a course that he/she has already made. The update is then reflected on the server as well.

Entrance.js

Serves as the main page of the website. This file is responsible for logging in the user or redirecting them to the sign-up page.

Header.js

Serves as the header(top bar) of the website once a user has logged in. The name “Quizzly” can be seen. This file is also responsible for calling the logout function if a user chooses to logout.

Layout.js

Serves as the file that layout the view for the user: such as the side tabs and the header. This file is important to making sure everything is where it should be.

Lecture.js

Serves as the lecture object for the professor to ask his quiz to. This file is no longer needed.

Lectures.js

Serves as a mechanism to bring all the lectures together. This file is no longer needed.

LecturePanel.js

This file is the lecture panel where the professor is able to add questions/quizzes to a specific lecture. This feature has been removed from Quizzly.

MetricData.js

Serves to retrieve the specific information that is needed by MetricModal. This will pull the scores and data for students.

MetricModal.js

Allows the professor to interact and select that data that they want to look at from the web.

MectricSection.js

Shows the professor attendance for a specific section.

MetricSectionQuiz.js

Shows the professor the quiz statistics based on section.

MetricSectionStudent.js

Shows the professor a student’s statistics overall.

MetricSectionStudentQuiz.js

Shows the professor a student’s statistics for a quiz.

Metrics.js

Serves as the mechanism to pull data from the server for metrics to function properly in the Metrics tab.

Profile.js

Populates the logged in user’s information in the ProfileModal file.

ProfileModal.js

Shows the information of the logged in user. The user can also edit their information from this view on the website.

Question.js

Serves as the question object that is asked to the students and created by the professor for a course.

Quiz.js
Serves as the quiz object. This quiz is for each course and then populated in each section.

Quizzes.js
Serves as a mechanism to bring together all quiz objects to display them for a specific section/course.

SectionBarChart.js
Grabs data to show student attendance per section in pie chart form.

SectionStudentBarChart.js
Grabs data to show student attendance per section in bar chart form.

SectionStudentPieChart.js
Grabs data to show student attendance per section in pie chart form.

SectionStudentQuizBarChart.js
Grabs data to show student sections for a quiz in bar chart form.

SectionStudentQuizPieChart.js
Grabs data to show student sections for a quiz in pie chart form.

Sidebar.js
Serves to add the functionality to the sidebar that is seen by the student or the professor.

Solution.js
Responsible for showing the student if they got the answer right or wrong by highlighting their answer green or red.

StudentList.js
Shows the students that are enrolled in a section for a course. Students can be added to this list by using their email.

StudentMetrics.js
Serves as the container for the students metrics. This file is not in use.

StudentQuestion.js
Serves to show the student if they got the question correct or incorrect.

StudentQuestionModal.js
Serves as the display that the student sees when they are logged into the website and taking a quiz. This file shows the student the question fullscreen.

StudentQuiz.js
Serves as the student quiz object that they have taken/or have yet to take.

StudentQuizzes.js
Serves as the mechanism that shows all the quizzes that the student has taken and/or has yet to take.

Style.js
Serves as the file that gives the Quizzly look to the webpage. This includes the green-blue color.