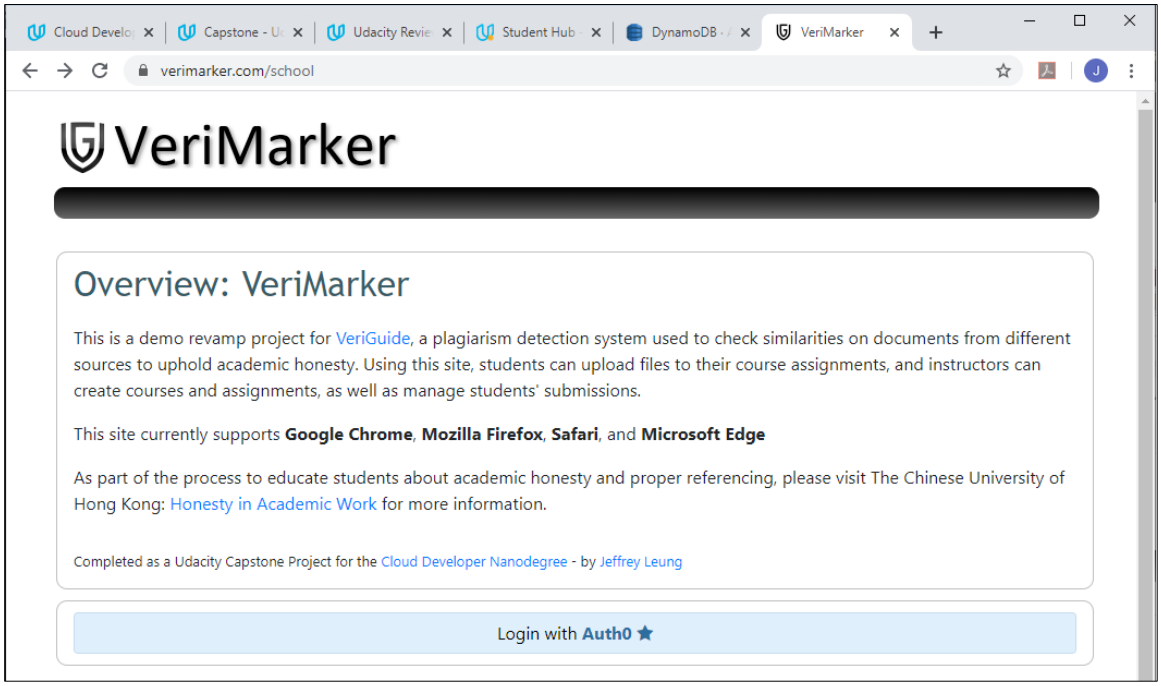
Contents

Cloud Developer Capstone Project - VeriMarker



1. Introduction

VeriMarker is a plagiarism detection system used by both students and instructors to manage submissions, assignments and courses, with the primary goal to ensure students did not plagiarized their work.

Using VeriMarker, student can submit files to the course assignments, view the similarity result, grades, and instructor's comments of their own submissions, and download the files of their own past submissions. Instructor can create and manage their own courses, assignments, and comment / grade the student's submissions uploaded to their own assignments. For the purpose of the capstone project, the similarity result of the student's submission is only a simulated number created by Math.random.

The Github repository of the project can be found in: https://github.com/jsleung1/project_capstone_verimarker and it also contains the instructors on how to setup the project.

1.1 Implementation

"Option 2" was selected for the Udacity Cloud Developer Capstone project. The backend was implemented using Serverless and AWS Lambda functions as taught in the project under Course 5 "Develop & Deploy Serverless App". The client is a Single Page Application written in Angular 8. The Angular client was deployed to AWS S3 / CloudFront with the help of Route 53. The deployed URL is https://www.verimarker.com

1.2 Demo

For demonstration purpose without setting up the Angular project, please go to https://www.verimarker.com and login via Auth0 using your Gmail account. You can also use the following Gmail accounts that are already registered in VeriMarker for a quick demo:

Gmail Login:	**VeriMarker Role**	**Password**
udacitystudent106@gmail.com	Student	EdTechFun101!
udacitystudent206@gmail.com	Student	EdTechFun101!
udacityteacher306@gmail.com	Instructor	EdTechFun101!

If you decide to use your own Gmail account, you will need to register in VeriMarker once you logged in from Auth0, in order to identify you as a Student or Instructor in VeriMarker. Subsequent logins will link your Auth0 token to your VeriMarker user account (stored in DynamoDB) in order to display the suitable menu functions for Student or Instructor. You can always change your user role to Student or Instructor by going to the "User Settings" in the main menu.

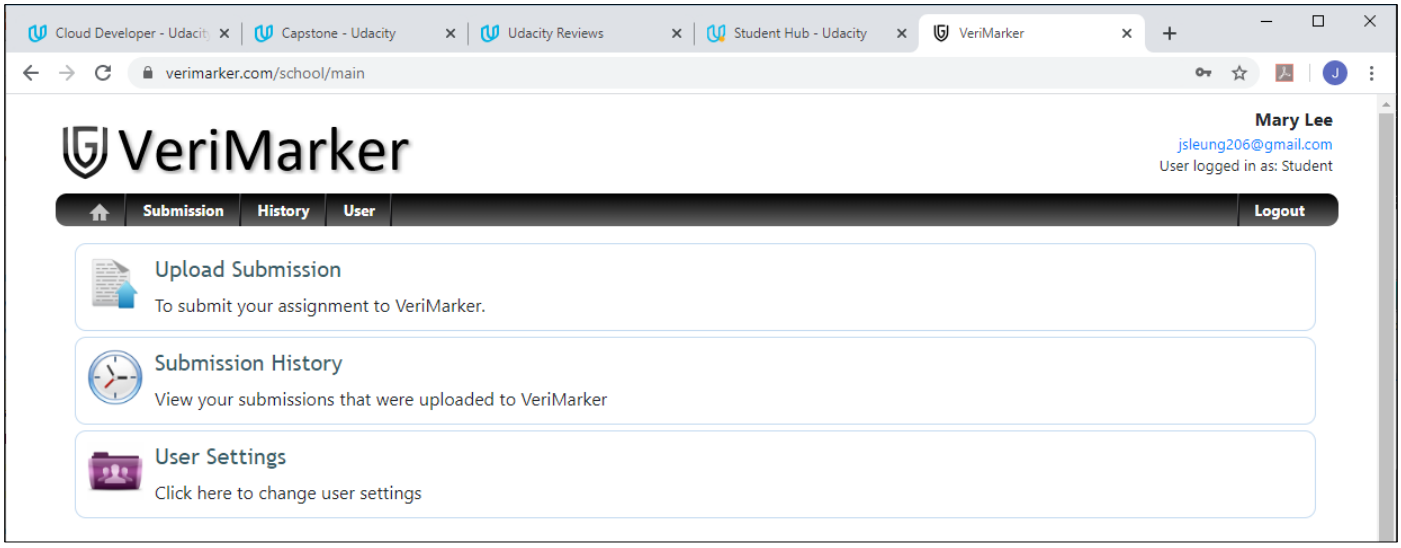
1.4 Trouble shooting

In the rare occasion that you receive an alert with message "invalid token error", please logout from VeriMarker. Another option is to delete the cookies in your Web Browser (it is caused when you logged in VeriMarker without logout the previous user).

2. Summary of User Functions

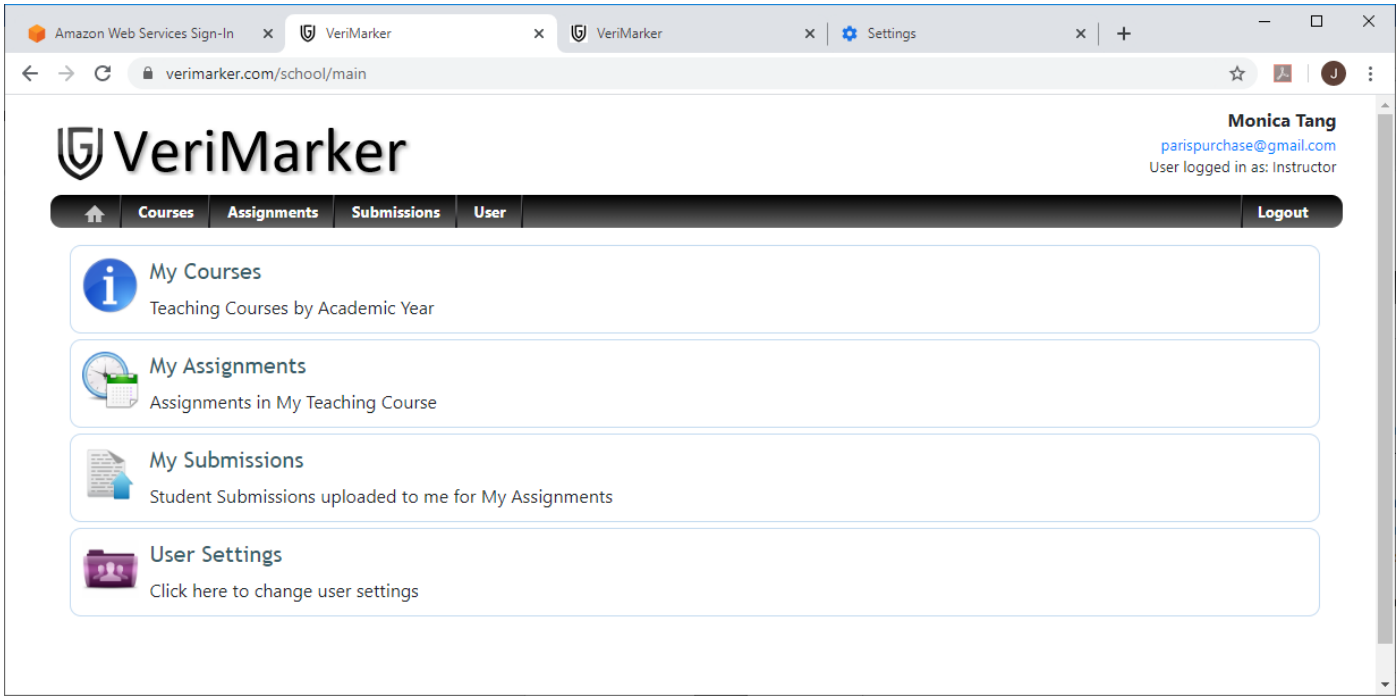
Before talking about VeriMarker is able to meet the project Rubrics in section 4, the following is a summary of the user functions of Instructor and Student. Please note that if this is the first time you logged in from Auth0 for the Gmail account, you will need to register as a new VeriMarker user as Student or Instructor.

2.1 Summary of user functions for the Student



Upload Submission	Student upload a file to the assignment, by select (in this order) the course, assignment and the instructor. Student must enter the Submission References (to indicate what references that they used to complete the assignment) before able to click on the "Upload Submission" button.
Submission History	Student can view their own submissions uploaded to the course assignments. Student can update the submission references which can be read by the instructor, and able to delete the submission (by click on the 'Withdraw button'). Once the submission is deleted, the instructor will not able to view the student's submission. Finally, in each submission, the student can view the similarity result, and the comments and score assigned by the instructor. The student can also download the file of the submission.
User Settings	Student can update the Email and change the user role (Student or Instructor) of their VeriMarker user account.

2.2 Summary of user functions for the Instructor



My Courses	Instructor can create courses (Student will able to view courses created by all the instructors).
	Instructor can view their own courses (Instructor cannot view courses created by other instructors.)
	Instructor can update the description of their own courses (Instructor cannot update courses created by other instructors.)
	Instructor can delete their own courses if it does not contain any assignments. (Instructor cannot delete courses created by other instructors.)
My Assignments	Instructor create assignment (by first selecting their own course).
	Instructor can view the assignments of their own course.
	Instructor can update the assignment description and assignment due date.
	Instructor can delete the assignment if the assignment does not contain any submissions.
My Submissions	Instructor can only view the students' submissions uploaded to their own assignments (Instructor cannot view the students' submissions uploaded to other instructors)
	Instructor can add comment and assignment score to the student submission.
	Instructor can download the file of the student submission.
	Instructor cannot delete the student submission (only student can delete their own submission by the 'Withdraw Submission' button).

	Instructor cannot update the Submission References of the student submission (only student can update the Submission References which is used by the Similarity Engine in the future).
User Settings	Instructor can update the Email and change the user role (Student or Instructor) of their VeriMarker user account. Please note the student will be unable to select the instructor from the dropdown in the submission upload screen if the instructor's user role was changed to a Student.

3. Unit Tests – Primary flow

The focus of this section is to present the Unit Tests of the primary flow described in the following table below. You may go directly to Section 4: Project Rubrics for meeting the project specifications. You may also find the Unit Tests already cover some of the requirements of the project Rubrics. Please click on the section link to go to the details of each unit test.

In DynamoDB, ensure Users, Courses, Assignments and Submissions Table are all empty.			
Section	**Unit Test Name**	**Unit Test Description**	**Expected Result**
3.1	Register new user as an Instructor	Register Gmail account (parispurchase@gmail.com) as Instructor, name = Monica Tang	Successfully registered the Gmail account as an Instructor in VeriMarker. Only Instructor functions are accessible by the user.
3.2	Instructor create course / update course description	Create following courses for Academic Year 2019: 2019-NURS-1151 Development of Nursing 2019-NURS-2112: Disaster Nursing 2019-WRONGCODE: Course to be deleted	Successfully created three courses under Academic Year 2019.
3.3	Instructor delete a course	Instructor delete course "2019-WRONGCODE: Course to be deleted"	Successfully deleted the course from the instructor.
3.4	Instructor create assignments	Create following assignments for 2019-NURS-1151: Assignment 1 Assignment 2 Create following assignments for 2019-NURS-2112: Assignment 1 Assignment X	Successfully created the assignments under each course.
3.5	Instructor delete assignment	Instructor delete "Assignment X" in course 2019-NURS-2112.	Instructor able to delete own assignment.
3.6	Register two new users as Students	Register Gmail account (jsleung206@gmail.com) as Student, name = Mary Lee Register Gmail account (jsleung106@gmail.com) as Student, name = David Liu	Successfully registered two Student accounts in VeriMarker. Only student functions are accessible by the user.
3.7	Student "Mary Lee" upload submission to the	Upload file to the following course / assignment: Course: 2019-NURS-1151: Disaster Nursing	File should be correctly uploaded to Assignment

	course assignment and check her submission history.	Assignment 1 Instructor: Monica Tang	1 in Course "2019-NURS-1151", to instructor Monica Tang.
3.8	Student "Mary Lee" upload the second submission to the same course and assignment.	Upload file to the following course / assignment: Course: 2019-NURS-1151: Disaster Nursing Assignment 1 Instructor: Monica Tang	Student's Submission History will show two submissions in Assignment 1 under Course "2019-NURS-1151"
3.9	Student "Mary Lee" deleted the second submission uploaded in previous section (3.8)	In submission History, student select the second submission, click on "Withdraw" submission to delete the submission.	The submission is deleted from the system (Student Mary Lee and Instructor Monica Tang will not able to view the deleted submission).
3.10	Student "David Liu" to upload submission to Assignment 1 for Course "2019-NURS-1151" and check submission history.	David Liu upload submission to the same course and assignment as in section 3.7 to 3.9.	The submission is successfully uploaded. Only one submission is listed in the Submission History of David Liu.
3.11	Instructor to check on student submissions uploaded to assignment 1 in section 3.7 to 3.10.	Instructor "Monica Tang" to check submissions uploaded by student "Mary Lee" and "David Liu" under Assignment of	Two submissions (one from each student) are found in Assignment 1 in Course 2019-NURS-1151. Instructor should able to download the file from both submissions.
3.12	Student update the submission references.	Student "Mary Lee" updates the references of her submission	Both Mary Lee and Instructor Monica Tang should able to view the updated Submission References.
3.13	Instructor add comments and score to the student's submissions.	Instructor Monica Tang add comments and assignment score to submissions of Mary Lee and David Liu.	Instructor able to update the submission with score and assignment comments. Students should able to view the instructor's score and comments.

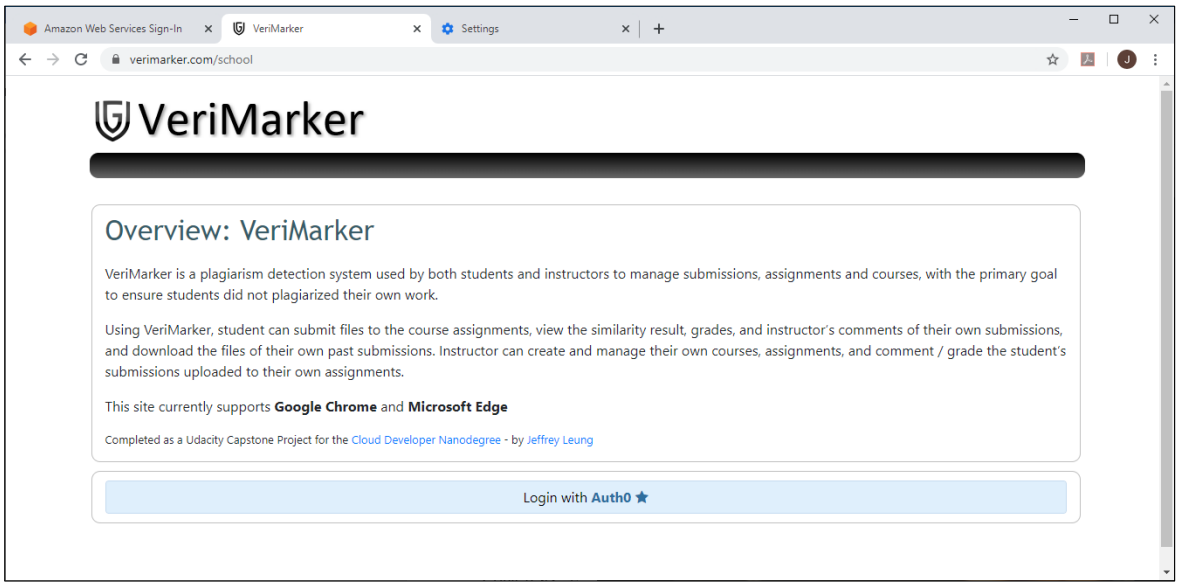
Initial data setup before the execution of the Unit Test:

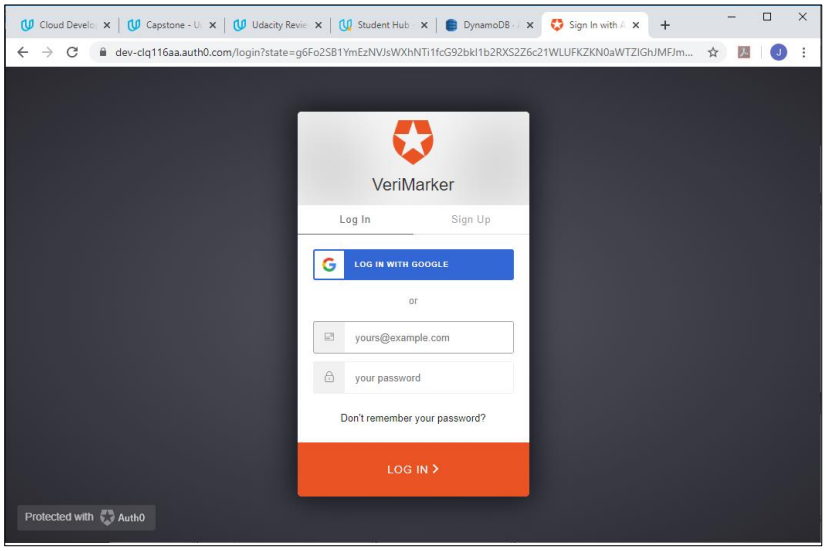
Professor Jeffrey Leung jsleung506@gmail.com Instructor	2019 Courses: 2019-ELTU-1001: Foundation English for University Studies Assignment 1 Assignment 2 Assignment 3 2019-ELTU-1002: English Communication for University Studies Assignment 1 Assignment 2 Assignment 3

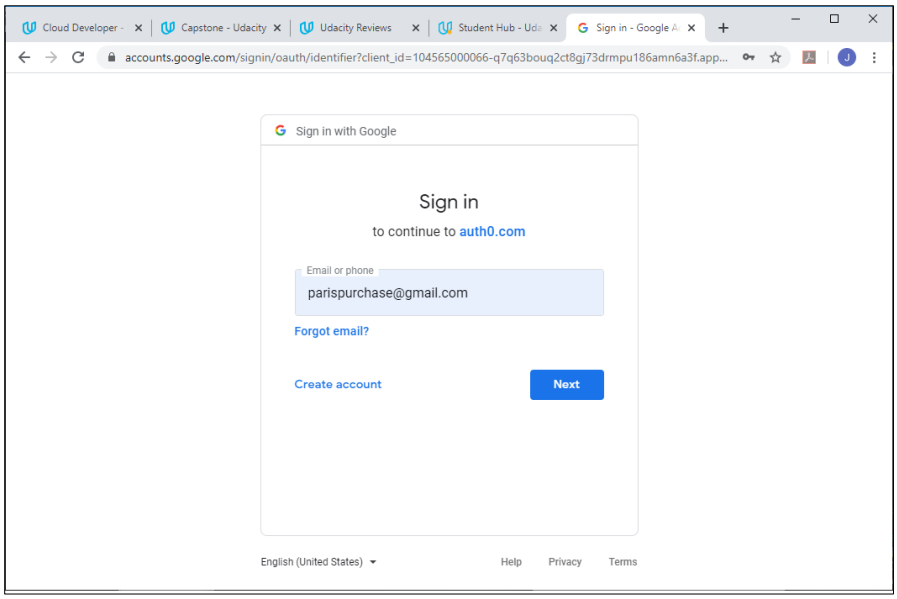
	2019-ELTU-2003: English Through Popular Culture Assignment 1 2019-ELTU-3011: English for Arts Students II Assignment 1 Assignment 2 Assignment 3 Assignment 4 2018 Courses: 2018-ELTU-1001: Foundation English for University Studies Assignment 1 Assignment 2 Assignment 3 2018-ELTU-1002: English Communication for University Studies Assignment 1 Assignment 2 Assignment 3 2018-ELTU-2202: Language Awareness for Teachers 1: Listening and Speaking Assignment 1 Assignment 2 Assignment 3
Dr. Karen Li jsleung406@gmail.com Instructor	2019 Courses: 2019-ACCT-3004: Accounting Practicum and Experiential Learning Assignment 1 Assignment 2 Assignment 3 Assignment 4 2019-ACCT-2111: Introductory Financial Accounting Assignment 1 Assignment 2 Assignment 3 2019-ACCT-1111 Foundations in Financial Accounting Assignment 1 Assignment 2

Section 3.1: Register new user as an Instructor:

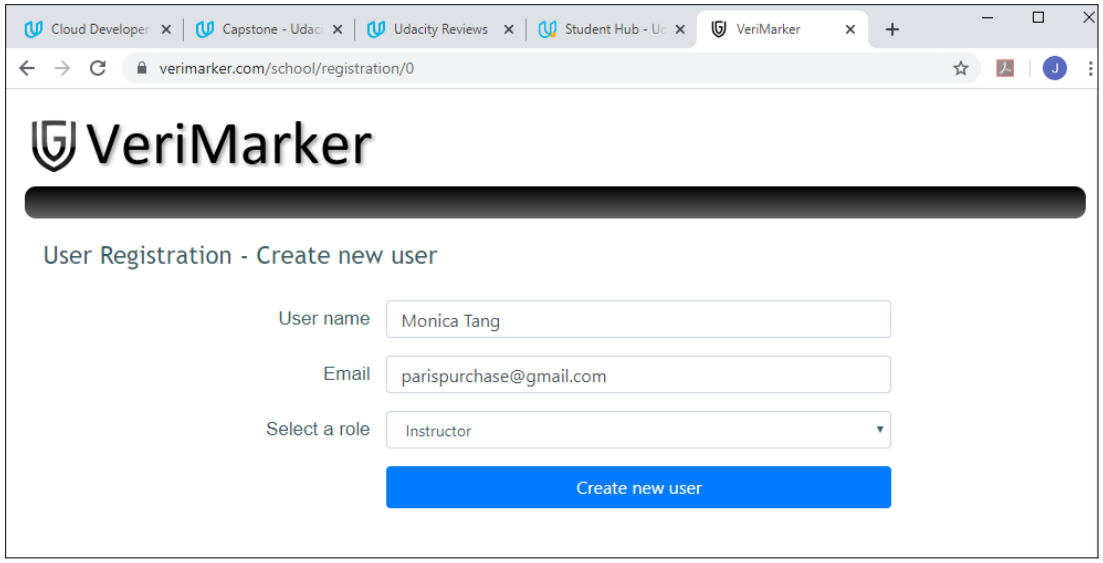
1. Logged in to VeriMarker using Auth0:

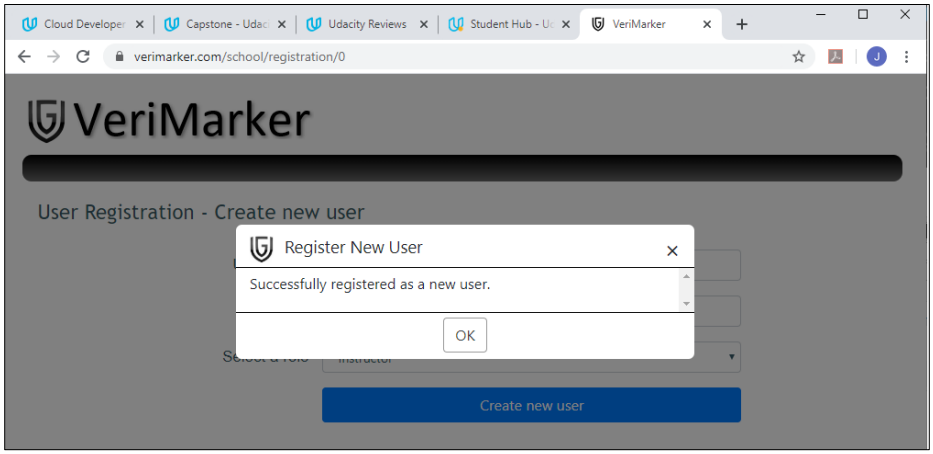






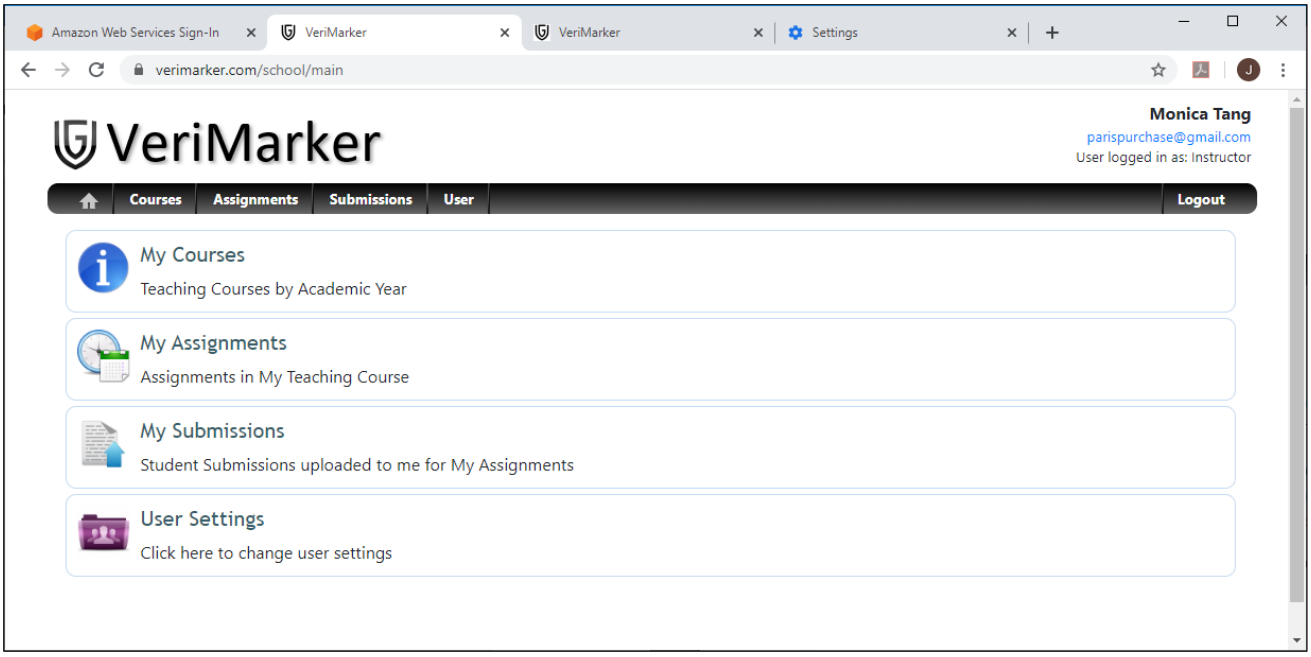
2. After signed in to Google, the user was redirected to the VeriMarker User Registration page. Enter the User name as Monica Tang, Email as parispurchase@gmail.com and select the role as Instructor, and click Create new user:





3. After the user is successfully registered, the user is navigated to the main menu of VeriMarker:

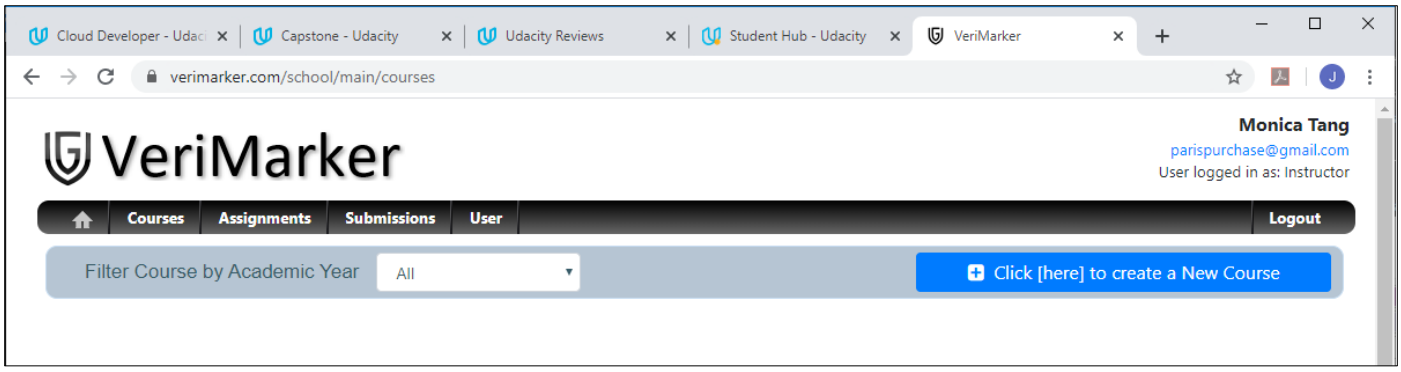
(continued to next page …)



4. User parispurchase@gmail.com was successfully registered as an Instructor with name Monica Tang. User is logged in to VeriMarker and the user can only access the Instructor functions from the main menu.

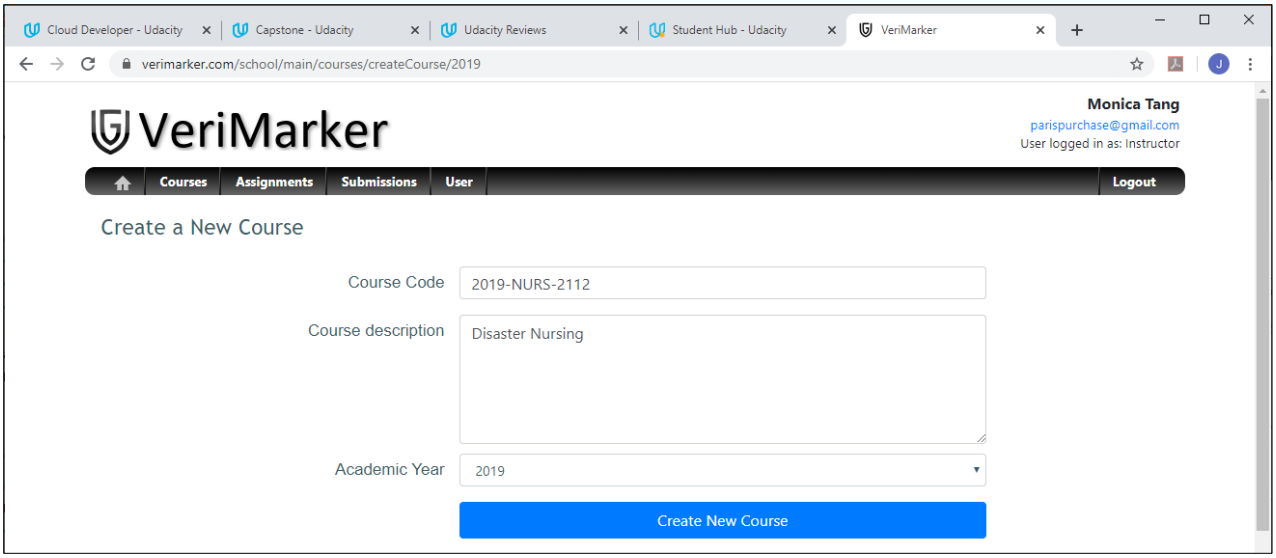
Section 3.2: Instructor create course / update course description

1. Click on the My Courses menu. It returns an empty list of courses. Click on the button to create a new course

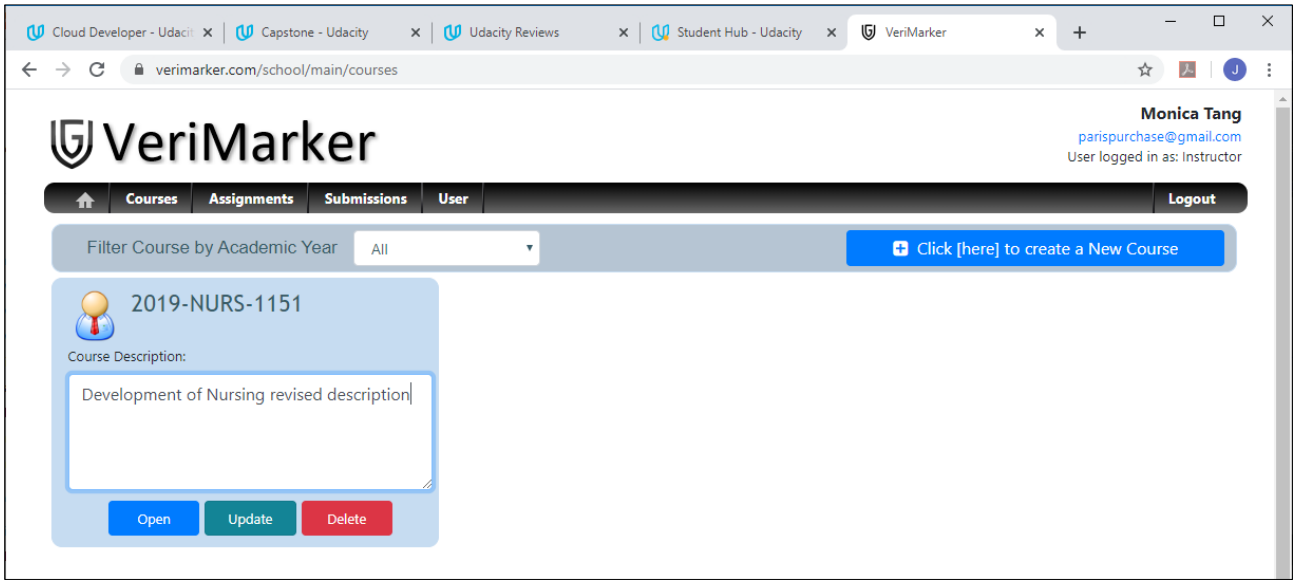


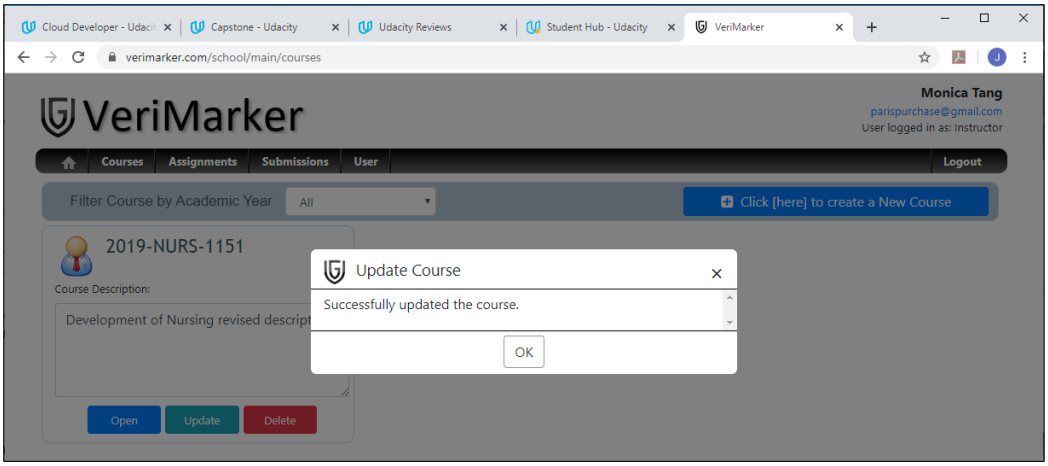
2. Enter the Course Code (must be unique for the selected Academic Year) and course description. Select the academic year (2019) for the course and click “Create New Course”:

(continued to next page …)

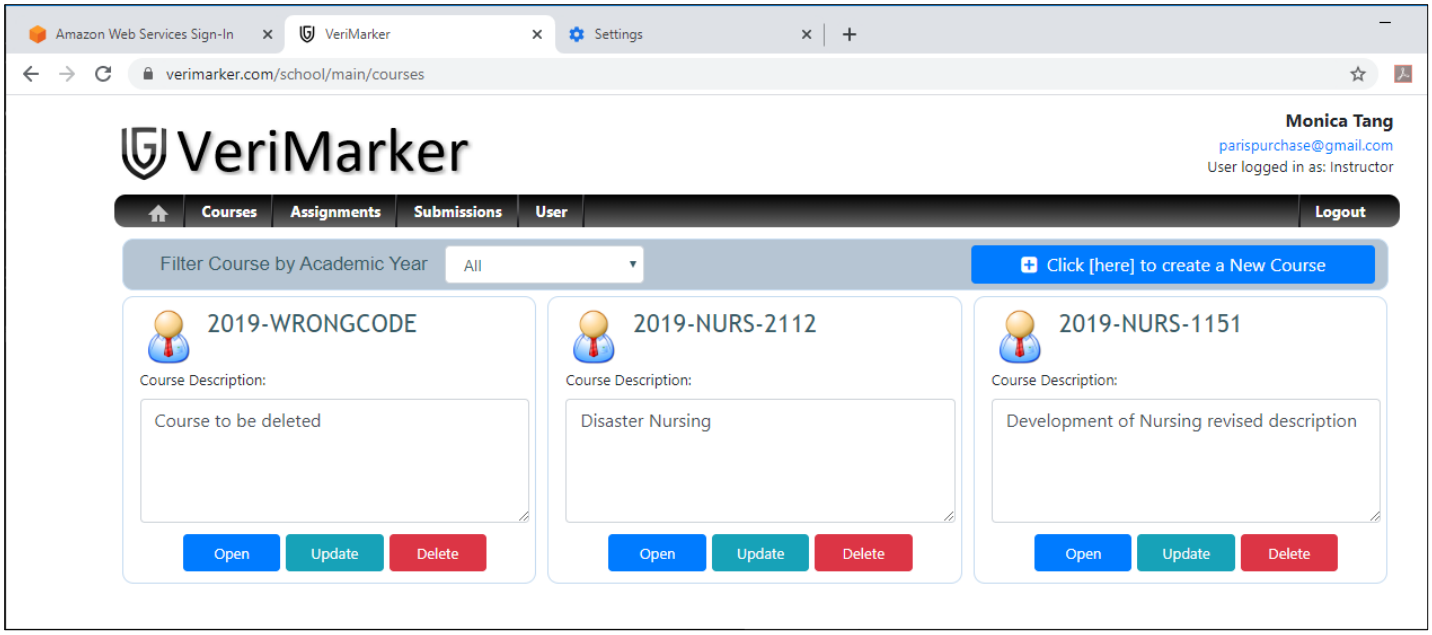


3. Instructor can now see the new course, and able to update the course description (by click on the update button):



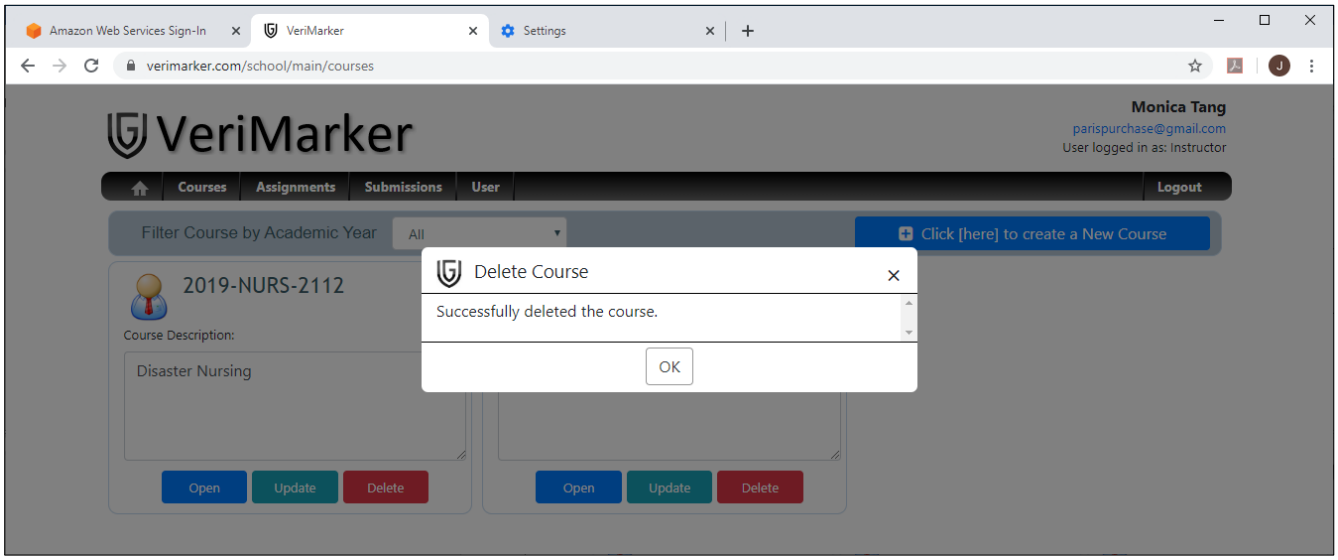


4. Repeat the above steps to create courses for “2019-NURS-2112: Disaster Nursing” and “2019-WRONGCODE: Course to be deleted”:



Section 3.3: Instructor delete a course

1. Refresh the Web browser. Click on the Delete button of the course "2019-WRONGCODE".

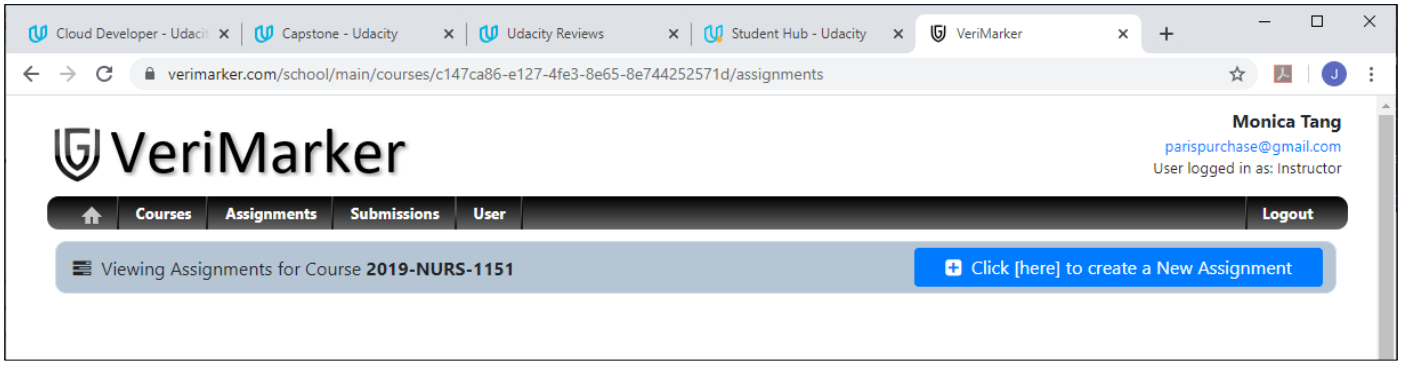


2. The course was successfully deleted and the instructor no longer able to view the deleted course.

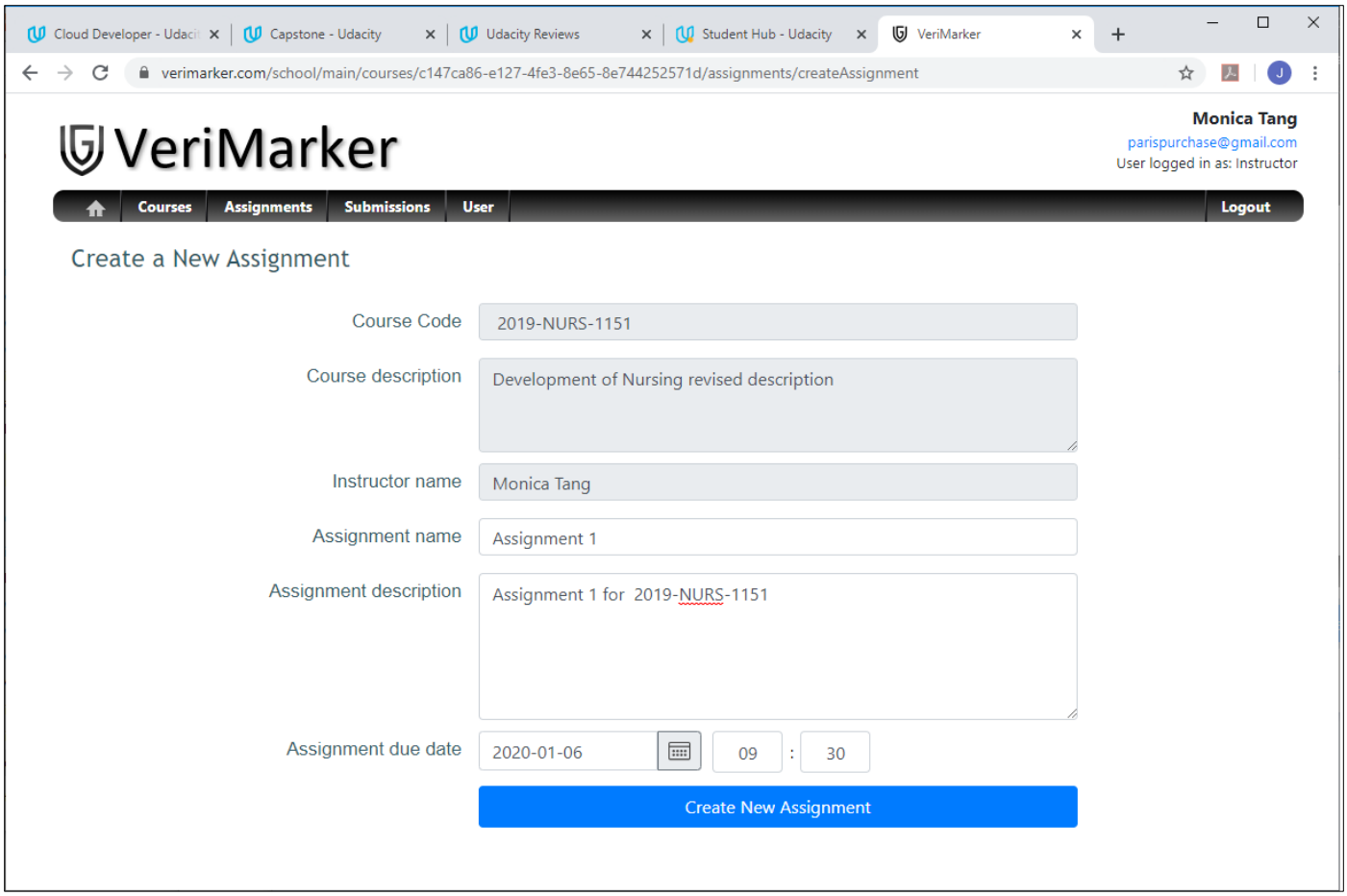
Section 3.4: Instructor create assignments

1. Instructor click on "Open" button of course 2019-NURS-1151. The page returns an empty list of assignments under the course as indicated below:

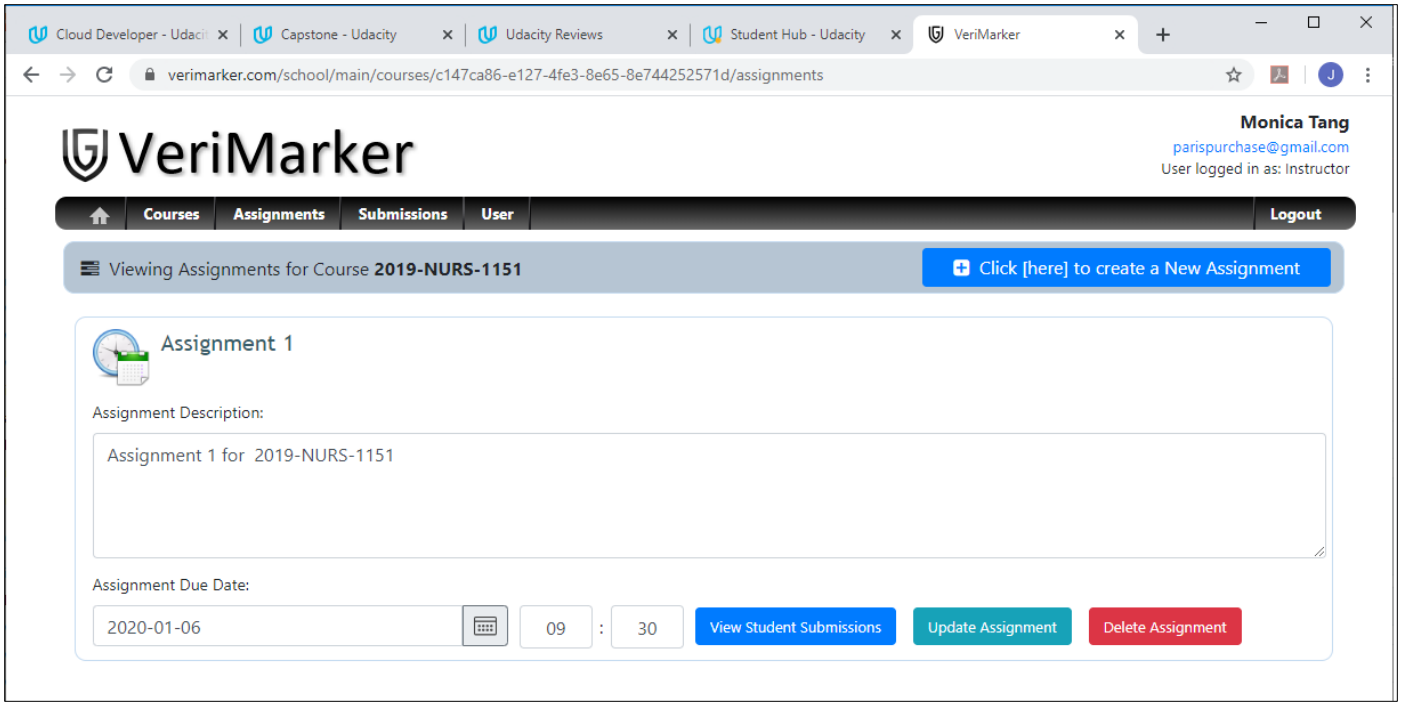
(continued to next page …)



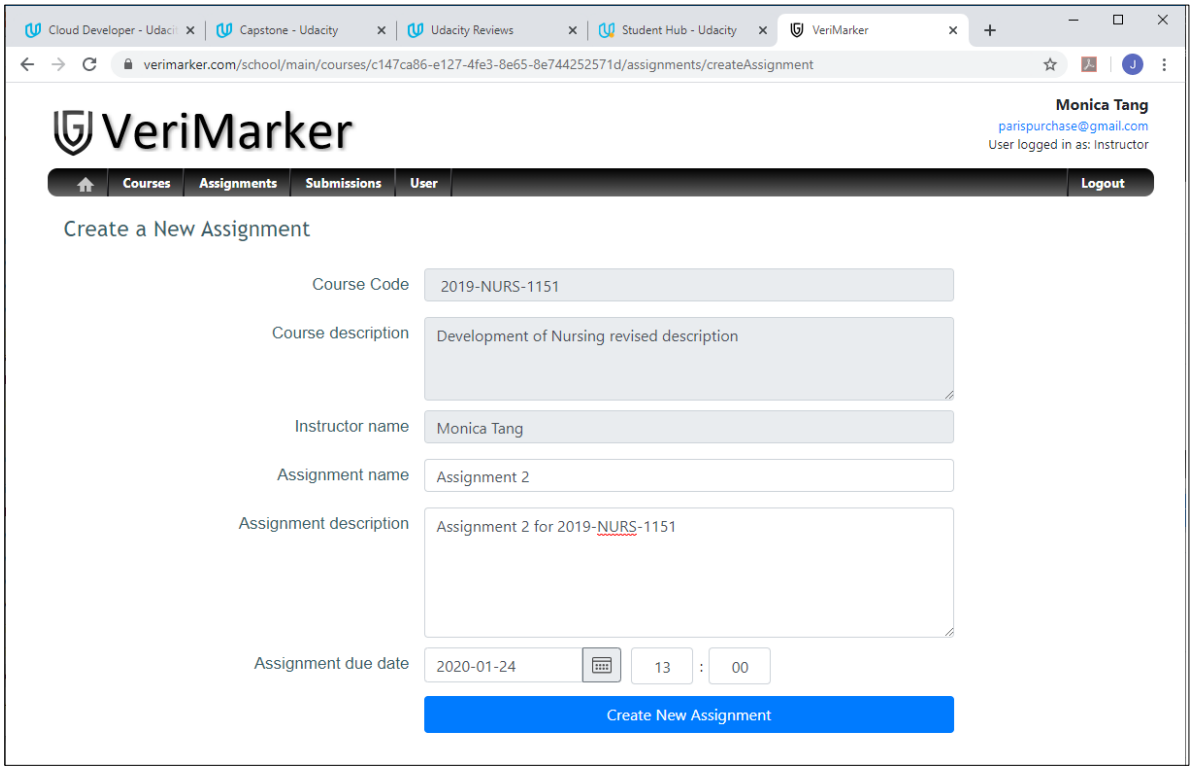
2. Instructor clicked on “Click [here] to create a New assignment”. Instructor enters “Assignment 1” as the assignment name and enter the description “Assignment 1 for 2019-NURS-1151’. Instructor should also specify the assignment due date (using “2020 January 6, 09:30” in the below example):



3. Instructor click “Create New Assignment”. The user successfully created the course. The user is redirect back to the list of assignments under course “2019-NURS-1151”, with the one assignment that was just created:

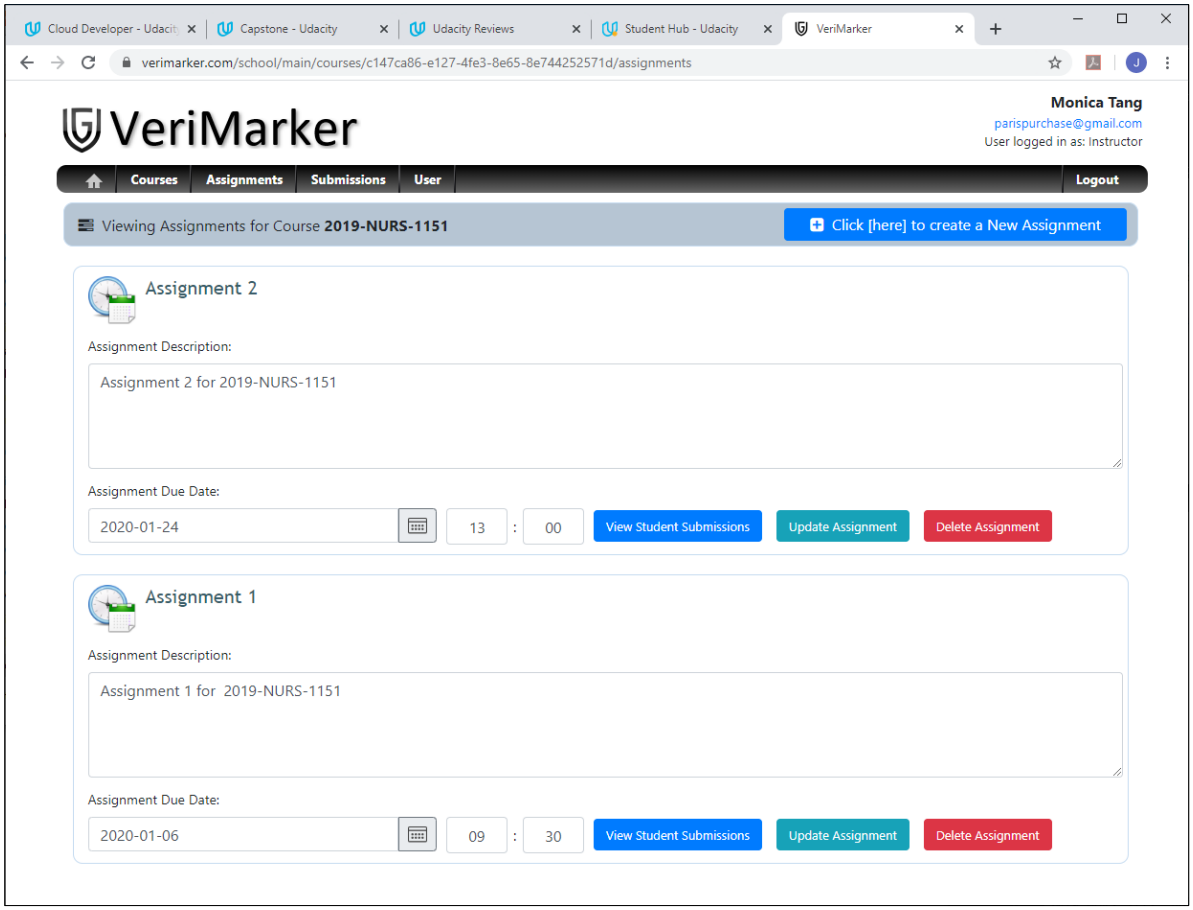


4. Repeat the above steps to create Assignment 2 under course “2019-NURS-1151”:

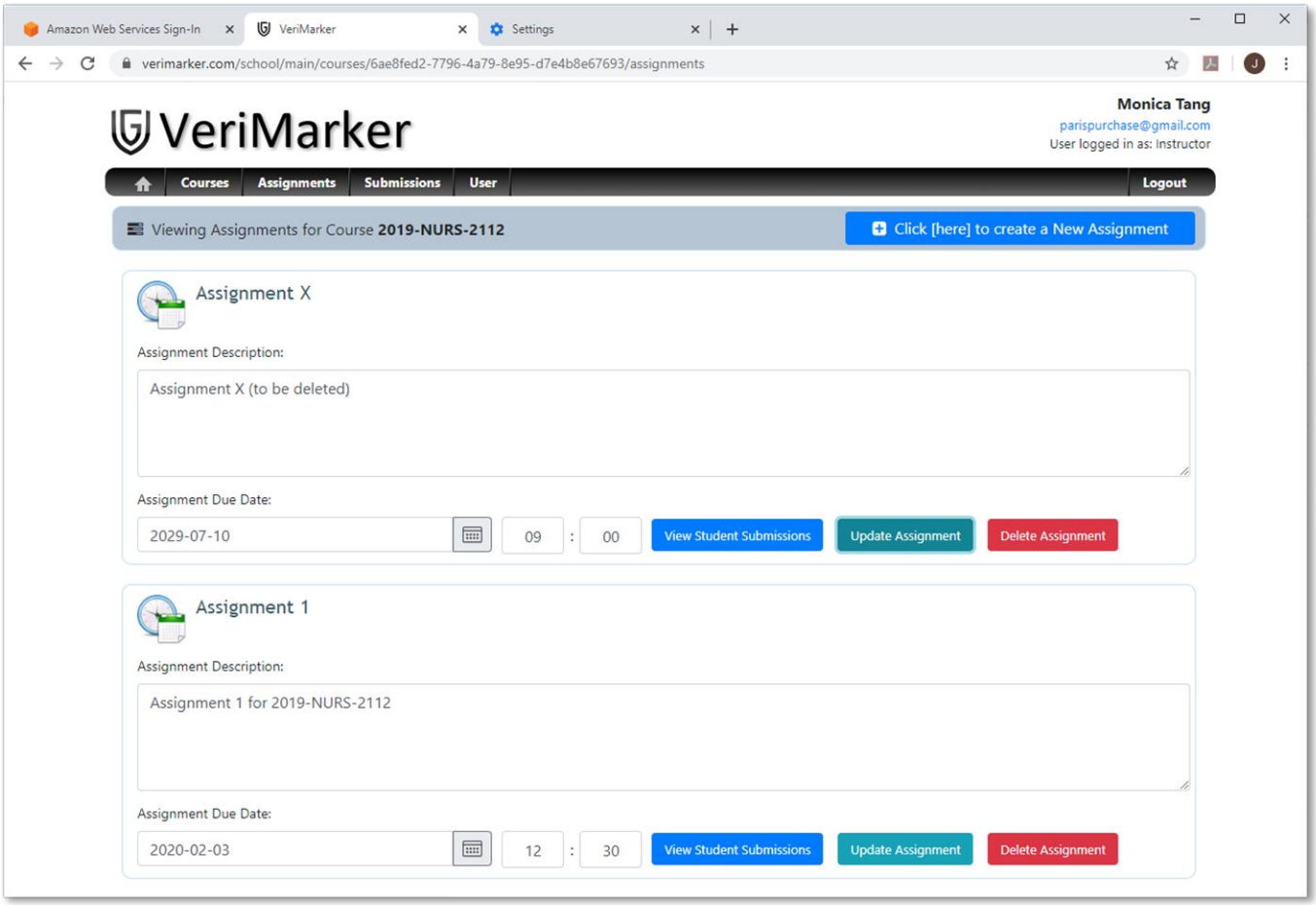


5. The list of assignments under course “2019-NURS-1151” will display Assignment 2 and Assignment 1:

(continued to next page …)



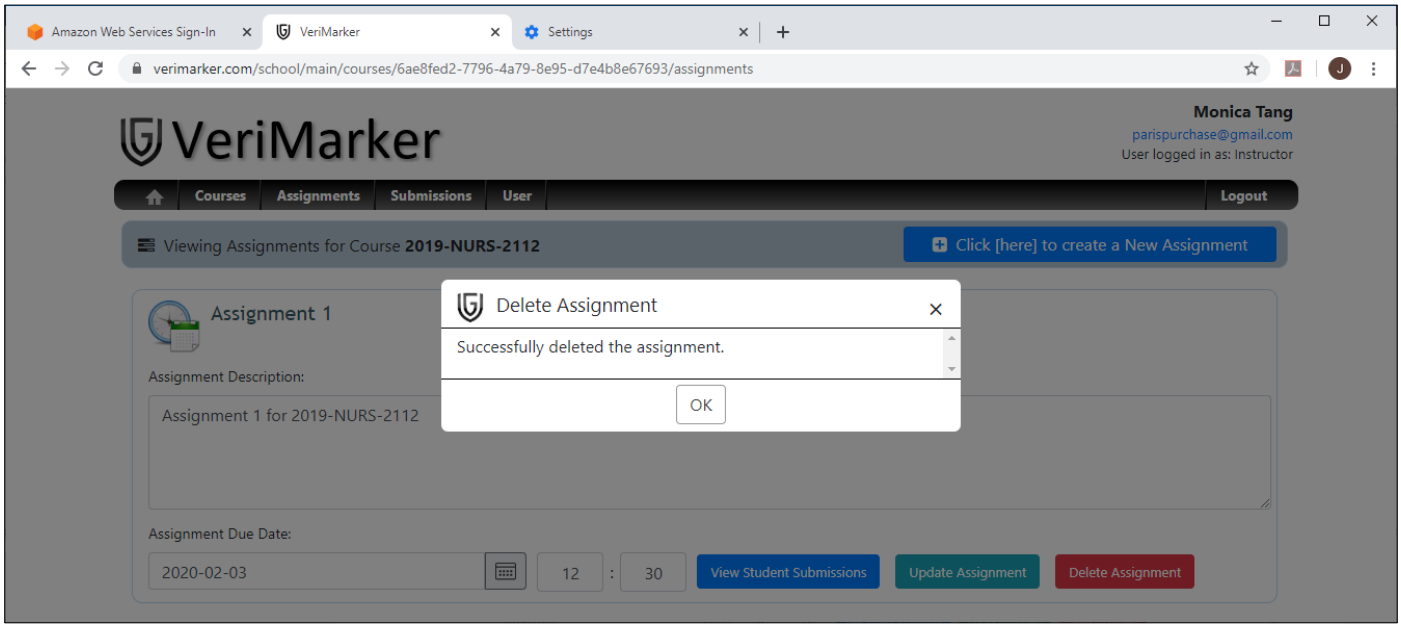
6. Repeat the above steps to create “Assignment 1” and “Assignment X” under course “**2019-NURS-2112**”:



7. The list of assignments under course “2019-NURS-2112” will display Assignment X and Assignment 1.

Section 3.5: Instructor delete assignment

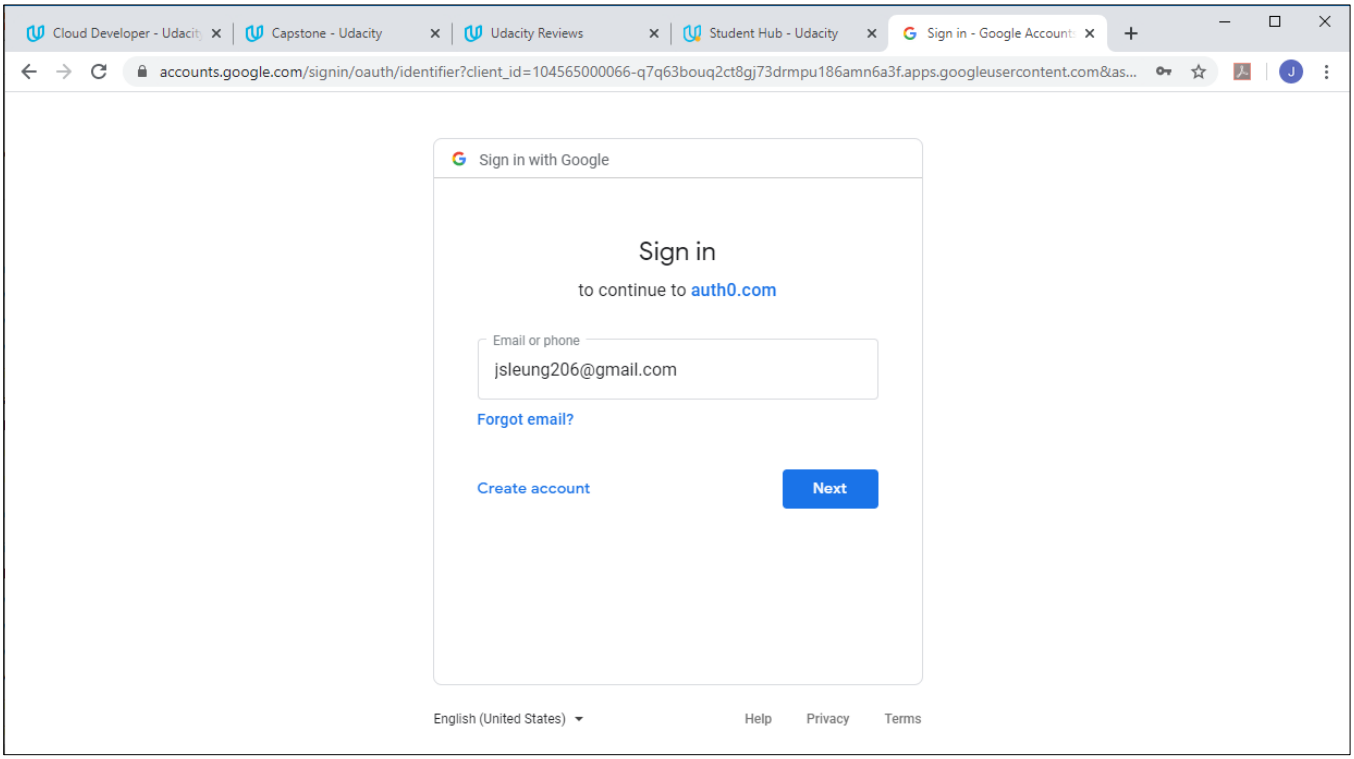
1. Click on the “Delete Assignment” button of “Assignment X” under course 2019-NURS-2112.



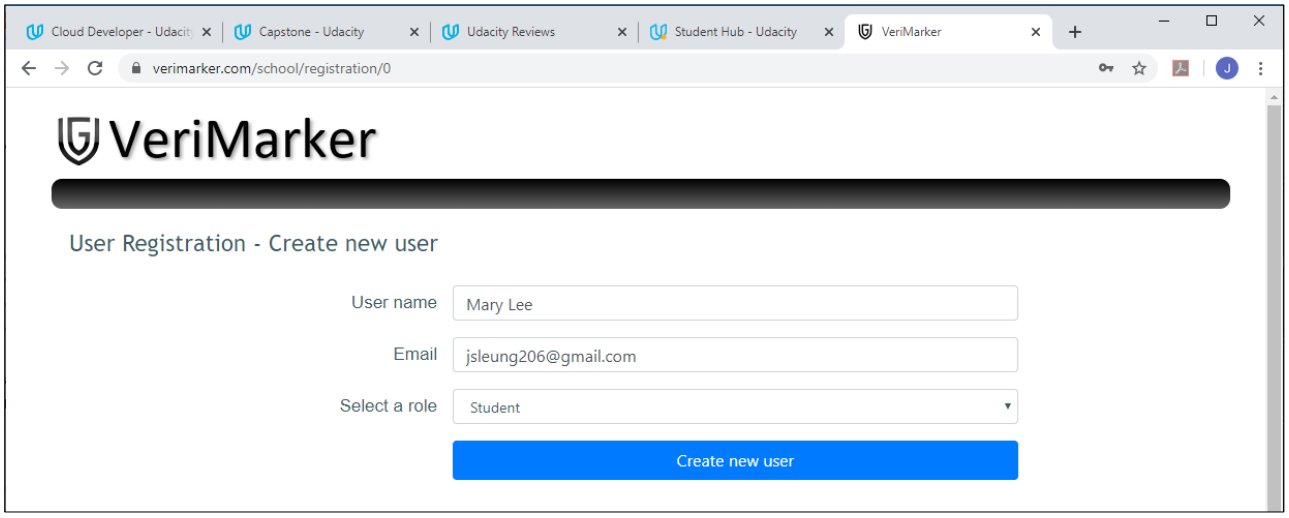
2. “Assignment X” was successfully deleted in Course 2019-NURS-2112. The instructor will no longer able to view the assignment “Assignment X”.

Section 3.6: Register two new users as Students

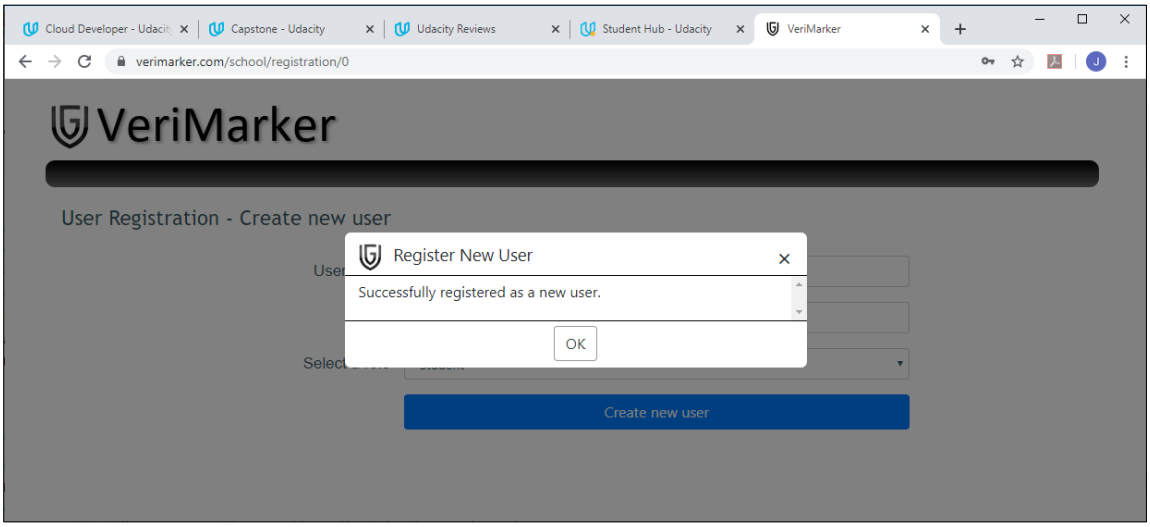
1. In VeriMarker, logout the instructor Monica Tang. Login with jsleung206@gmail.com to register the Google account as a Student in VeriMarker:



2. Enter the user name as Mary Lee, Email as jsleung206@gmail.com and select the role as Student, and click Create new user:

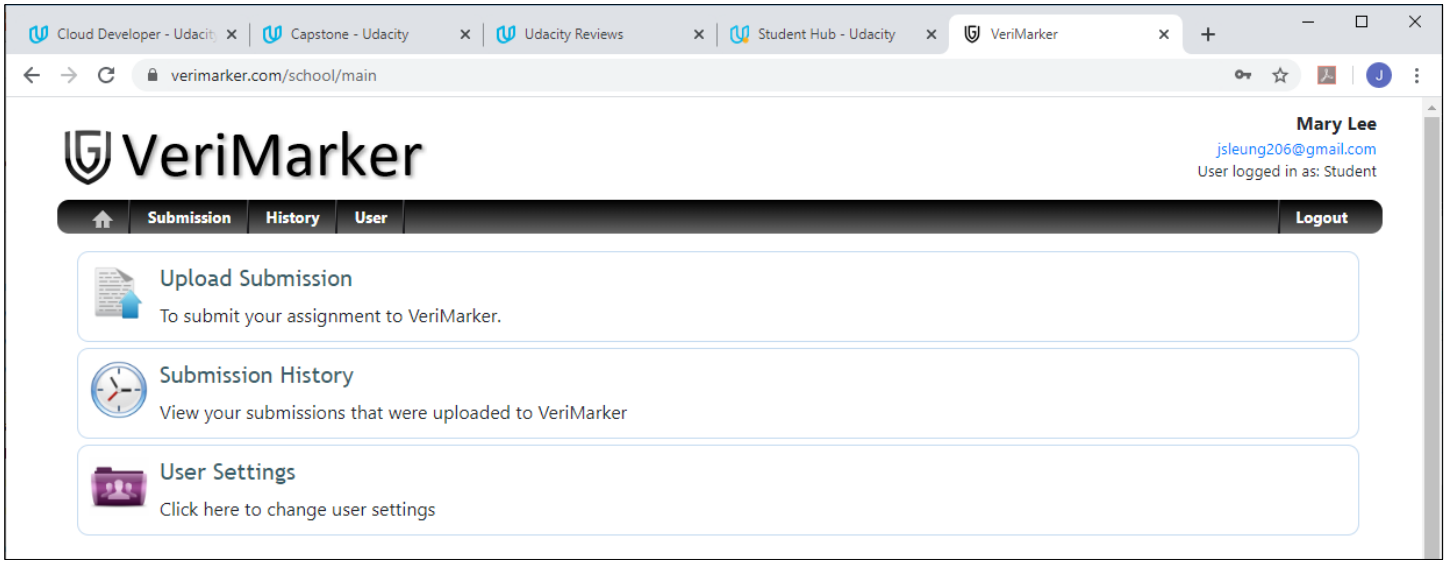


3. Enter the user name as Mary Lee, Email as jsleung206@gmail.com and select the role as Student, and click Create new user:



4. After user is successfully registered, the student is navigated to the main menu of VeriMarker:

(continued to next page …)



5. User jsleung206@gmail.com was successfully registered as a student with name Mary Lee. Mary Lee can only access the Student functions in the main menu.

6. In VeriMarker, logout the student Mary Lee. Repeat the above steps to register jsleung106@gmail.com as a new Student account with name "David Liu" in VeriMarker.

7. User jsleung106@gmail.com was successfully registered as a student with name David Liu. David Liu can only access the Student functions in the main menu.

8. Finish this section by logout David Liu in VeriMarker.

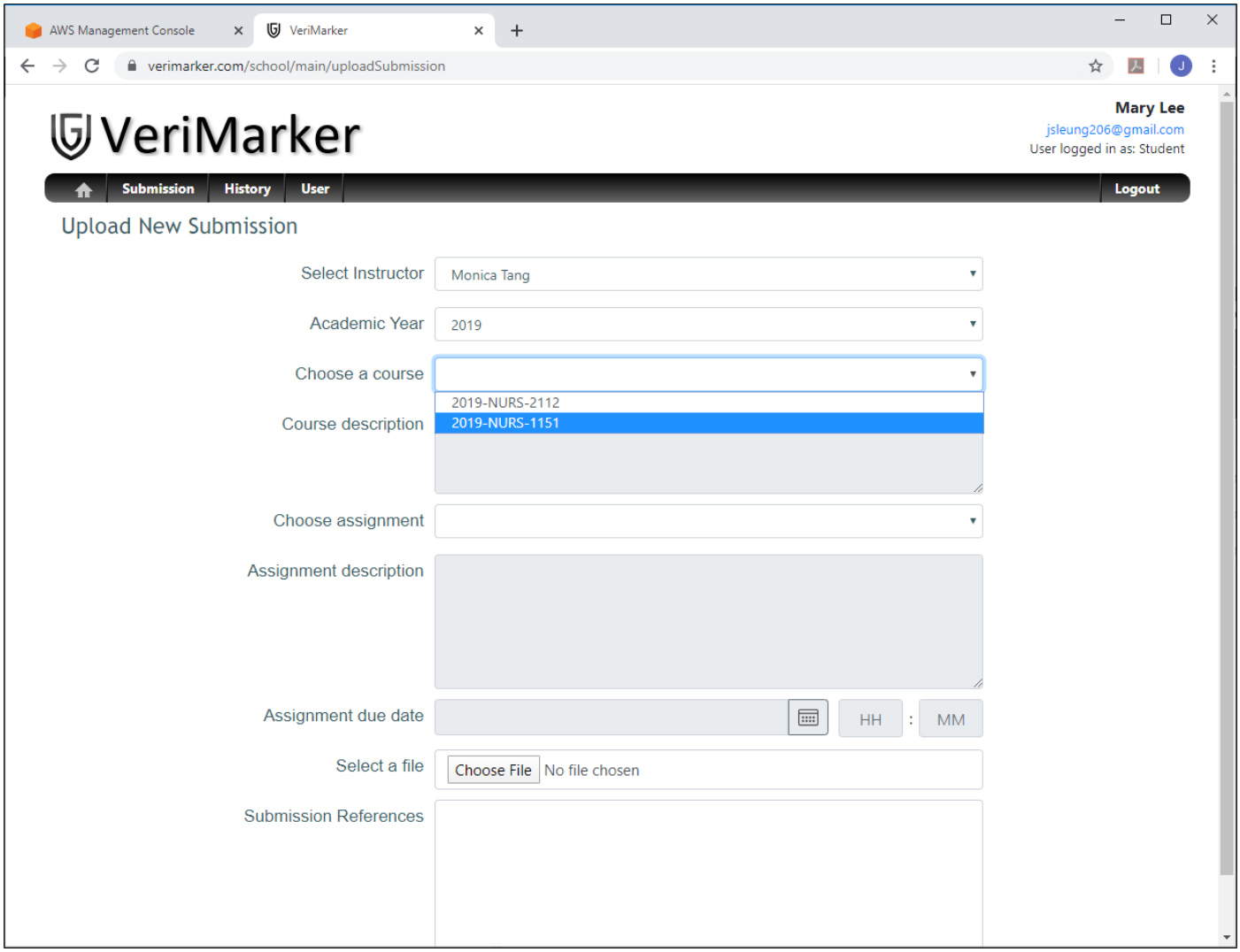
Section 3.7: Student "Mary Lee" to upload submission to the course assignment and check her submission history.

1. Login as Mary Lee using jsleung206@gmail.com.

2. Select Upload Submission in the main menu.

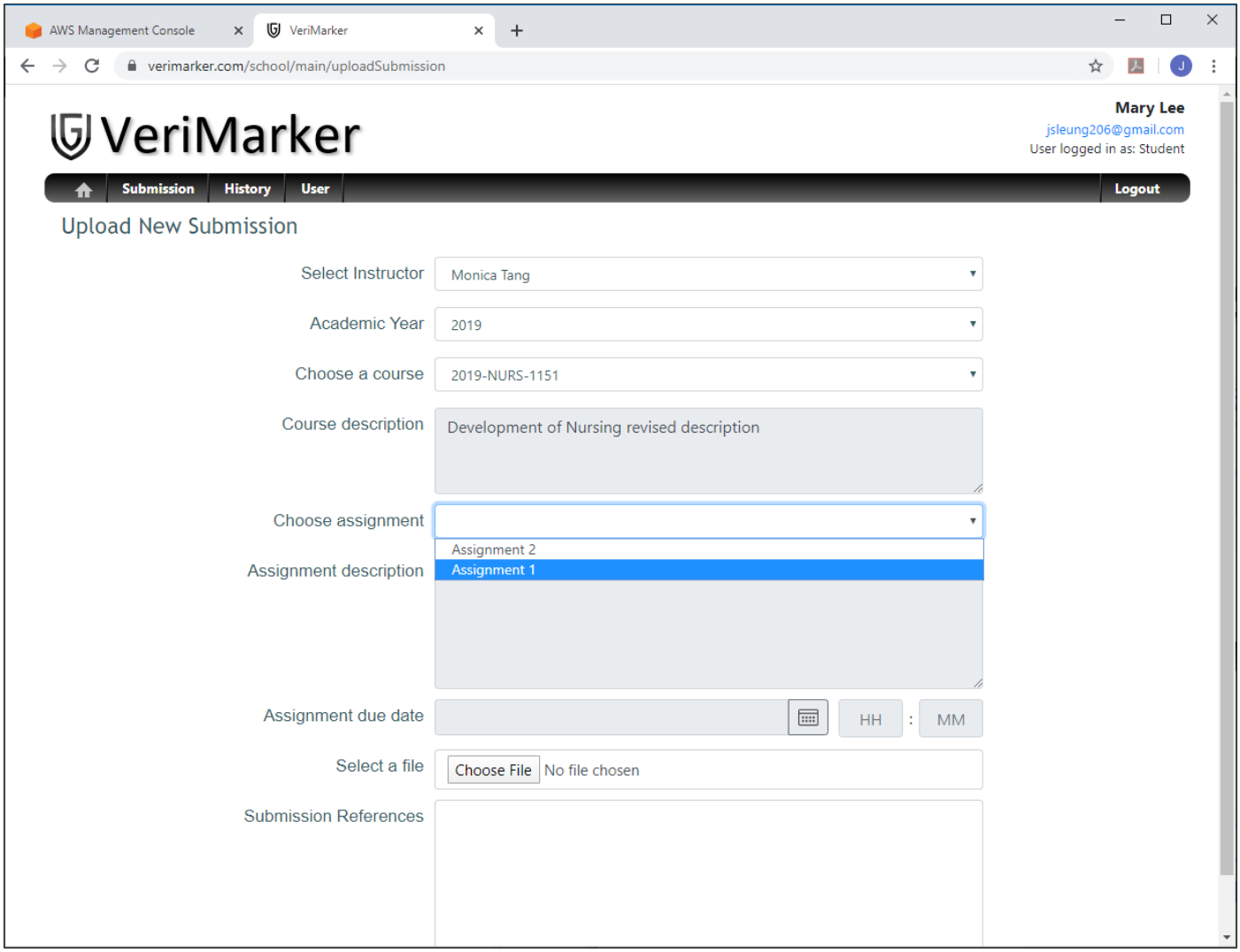
3. In the Upload New Submission page shown below, the student selects Instructor "Monica Tang", Academic Year = 2019, and choose a course "2019-NURS-1151" in the selection drop down:

(continued to next page …)



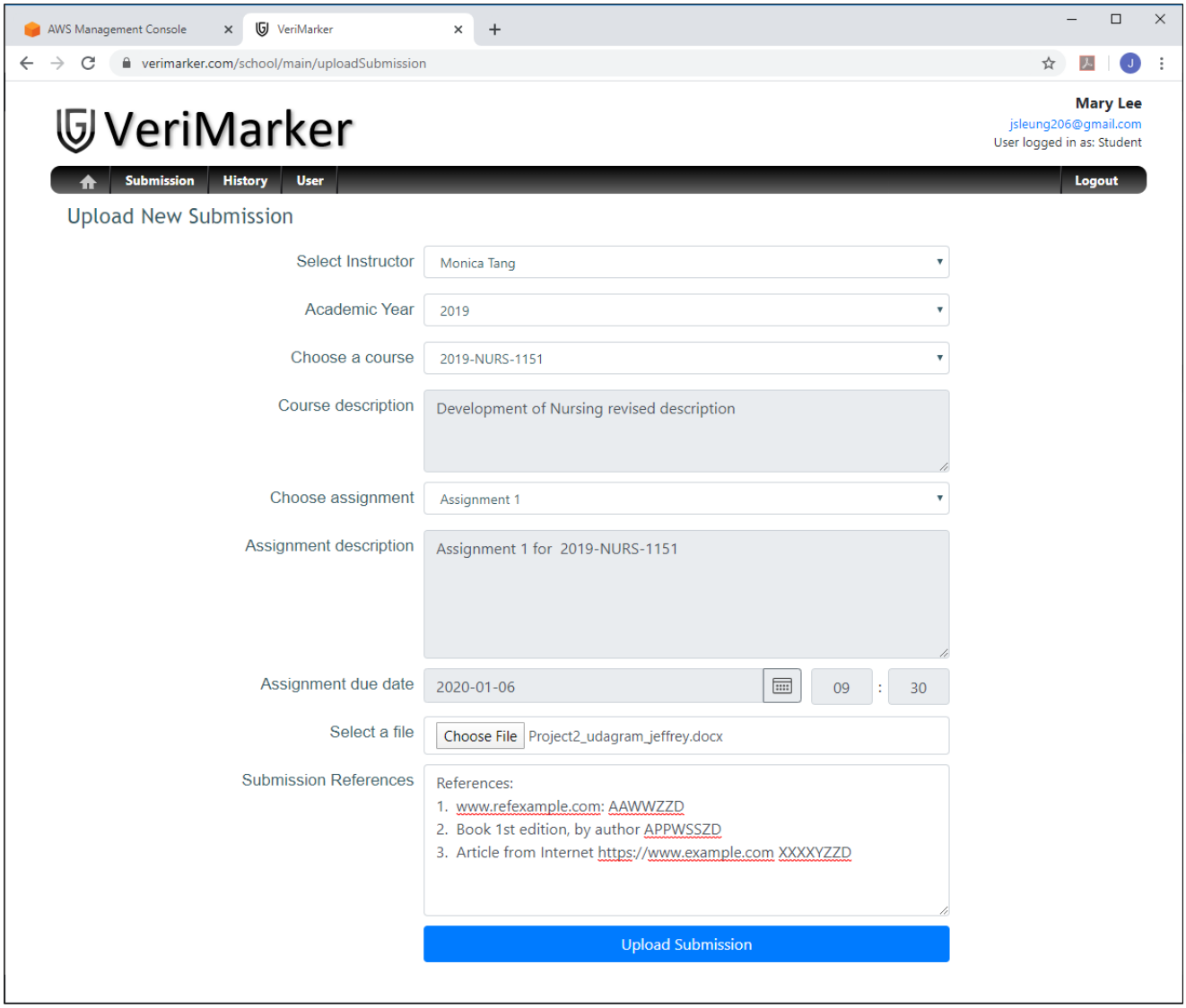
4. The correct course description should be displayed after the Course ("2019-NURS-1151") is selected. Student select Assignment 1 in the Choose Assignment drop down:

(continued to next page …)

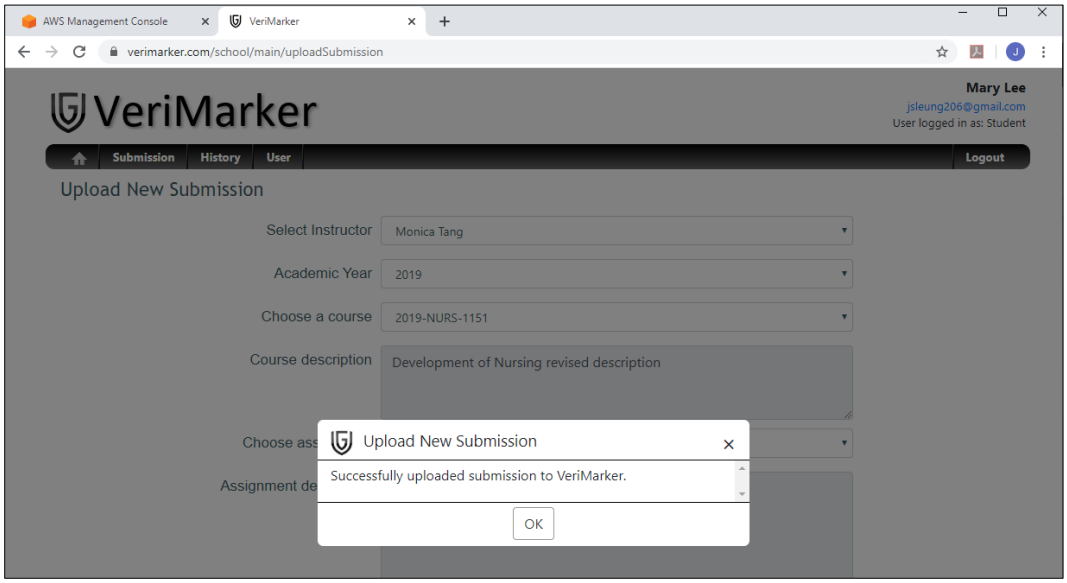


5. After the student selected “Assignment 1”, the correct due date “2020 January 6, 09:30” should be displayed. Student click on “Choose a file” to select the file “Project2_udagram_jeffrey.docx”. Student must enter the Submission References before able to click on the “Upload Submission” button:

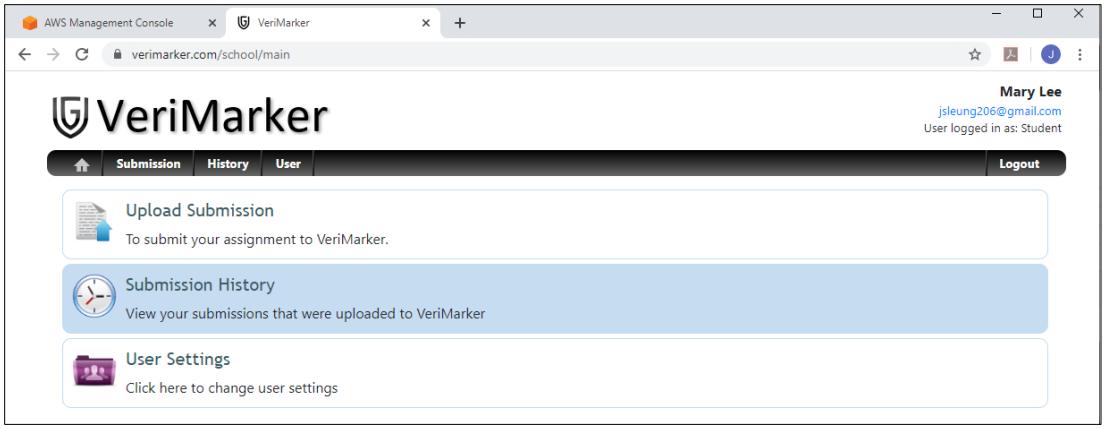
(continued to next page …)



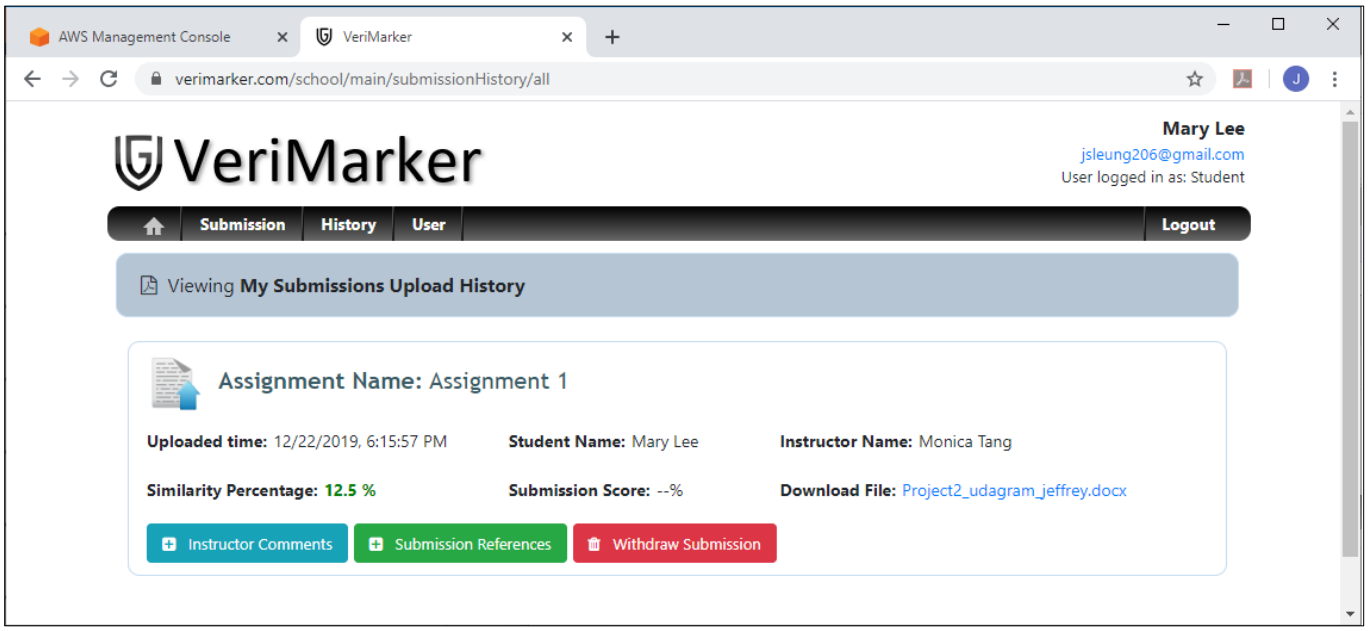
6. Student Mary Lee successfully uploaded the submission to VeriMarker.



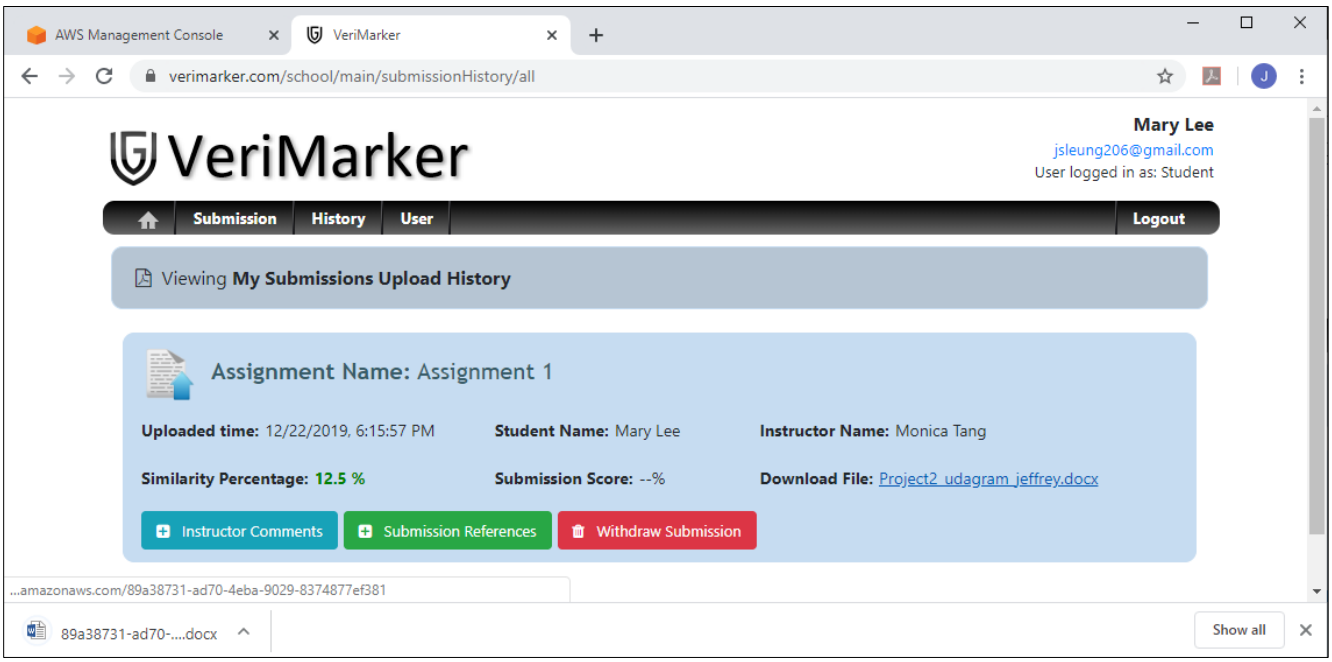
7. The student will be navigated back to the main menu. Select Submission History in the main menu to verify the submission was uploaded correctly

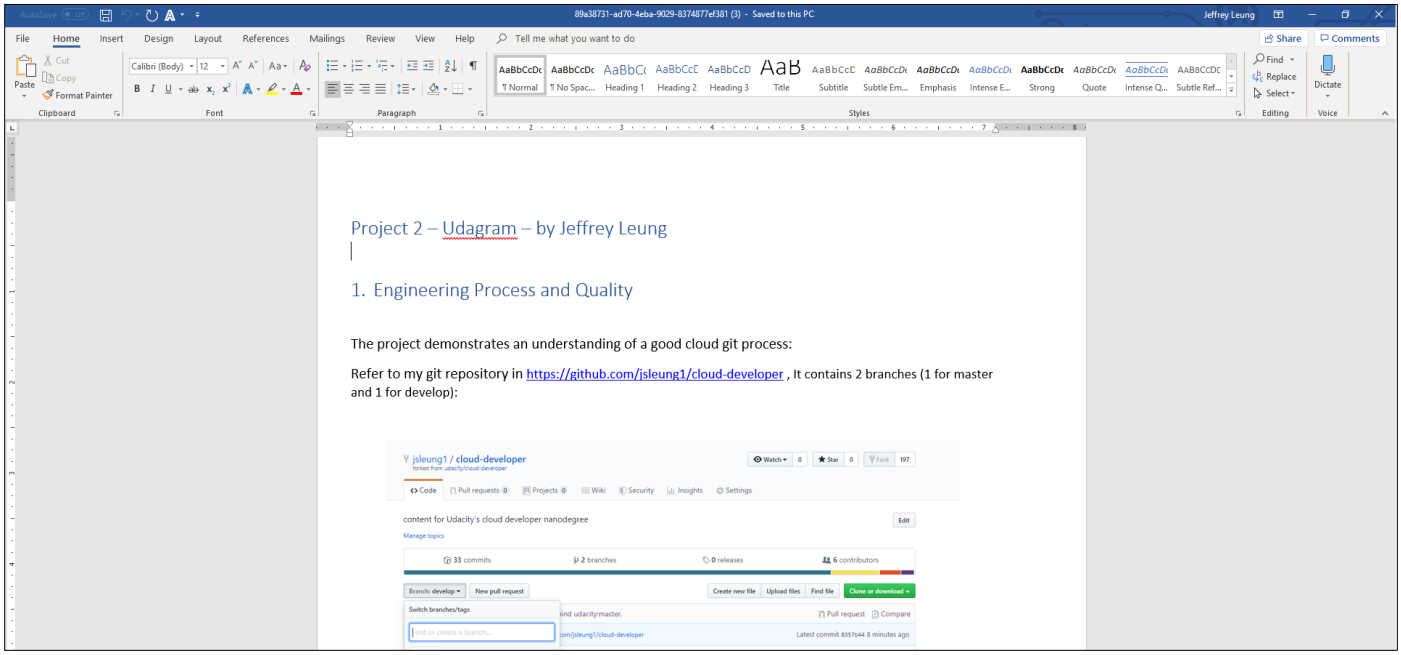


8. After click on Submission History, it will list the submission that was just uploaded:



9. Student click on the file name to start downloading the file. Open the file to verify the file contents are correct:

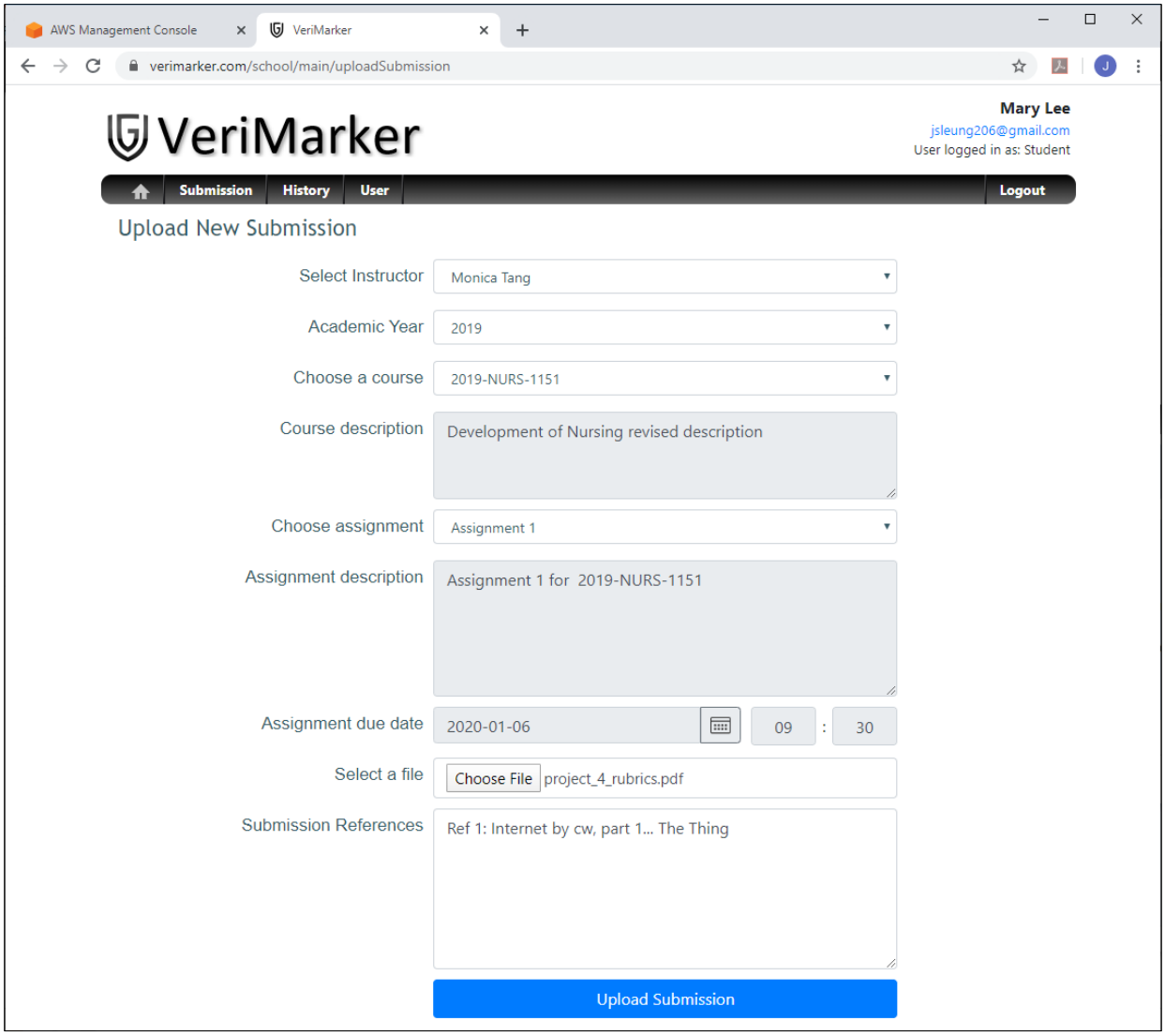




Section 3.8: Student "Mary Lee" upload the second submission to the same course and assignment.

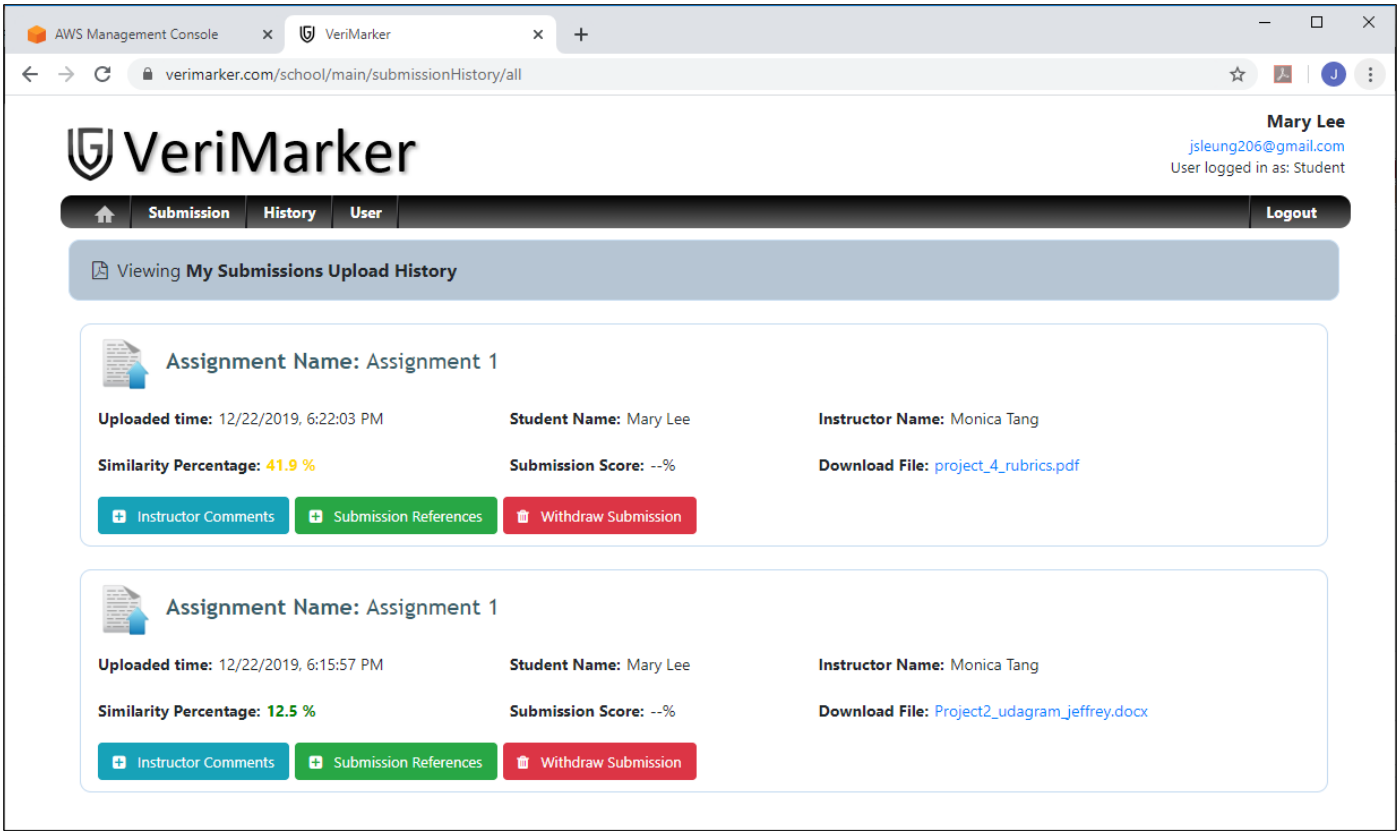
1. Repeat step 3.7 with a different file (project_4_rubrics.pdf):

(continued to next page …)



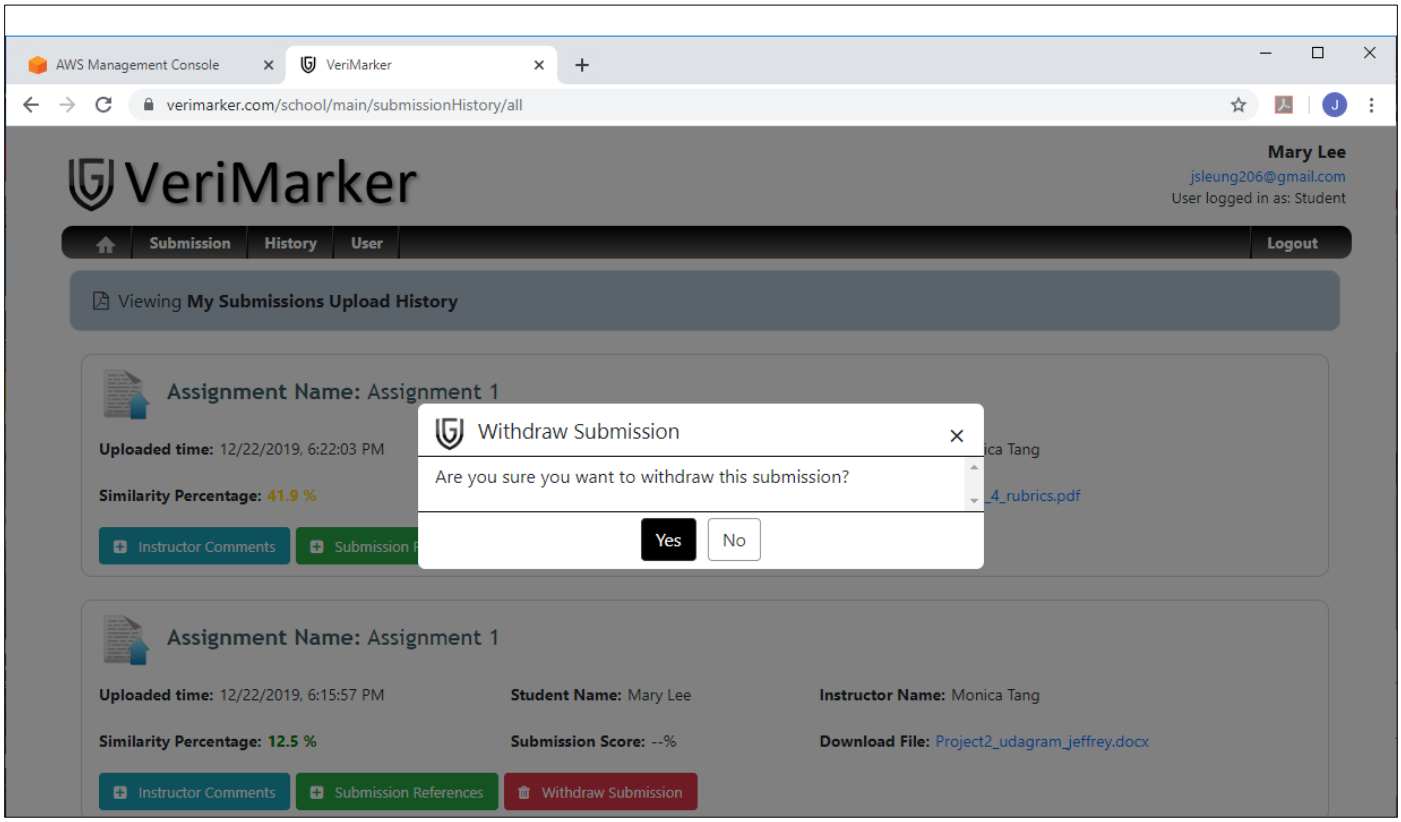
2. The submission history of student Mary Lee will display two submissions uploaded to Assignment 1:

(continued to next page …)

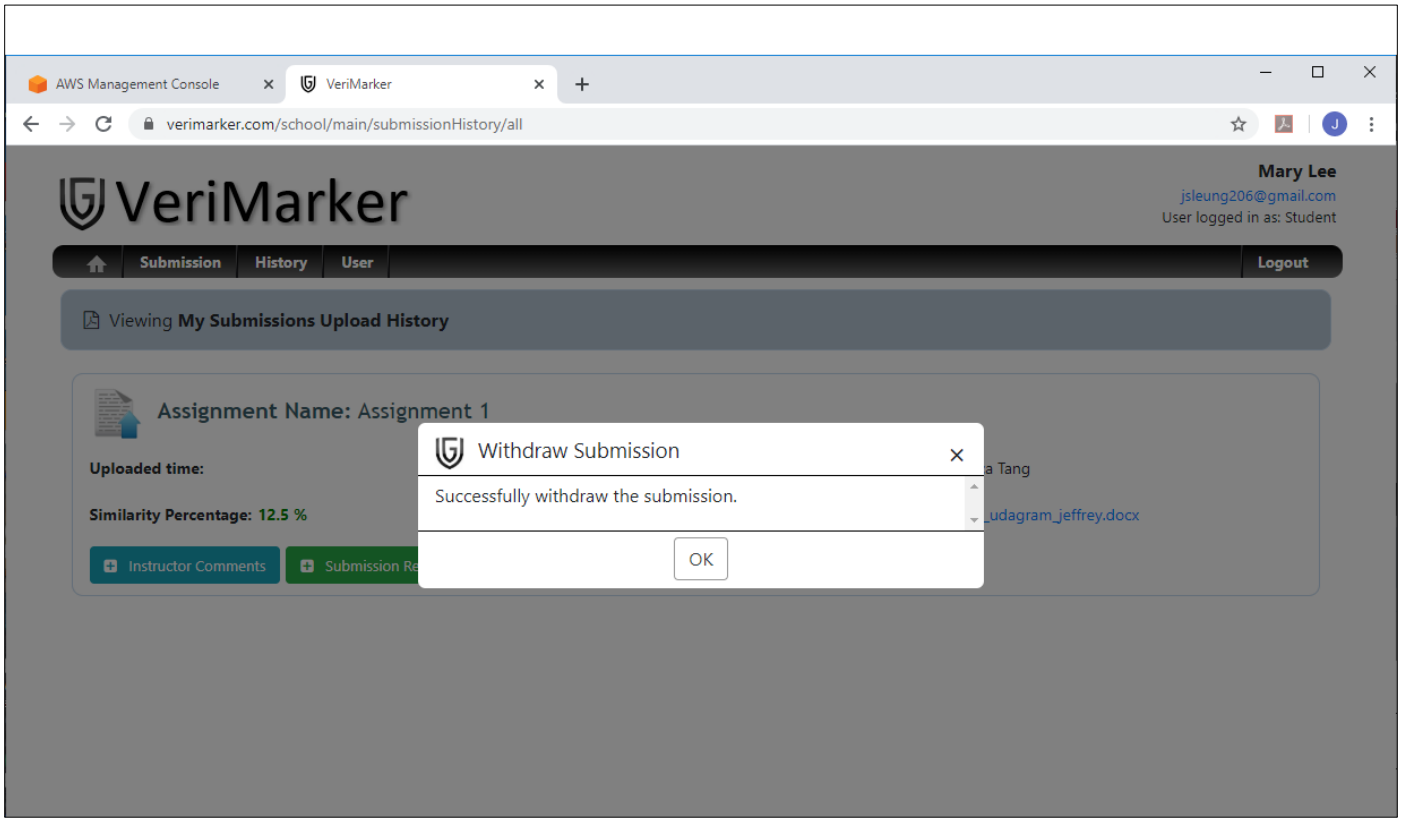


Section 3.9: Student “Mary Lee” deleted the submission uploaded in previous section (3.8)

1. In the submission history page, Mary Lee clicks on “Withdraw” submission button of the second submission (which is displayed first as the most recent submission):

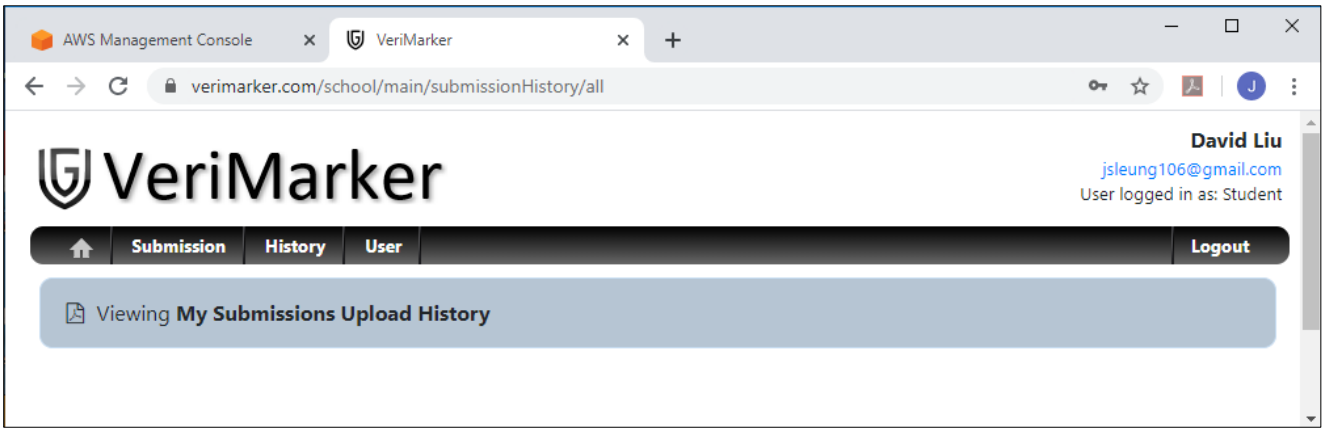


2. Click Yes to delete (withdraw) the submission. The second submission was deleted in VeriMarker, and the student is left with the first submission uploaded to Assignment 1:



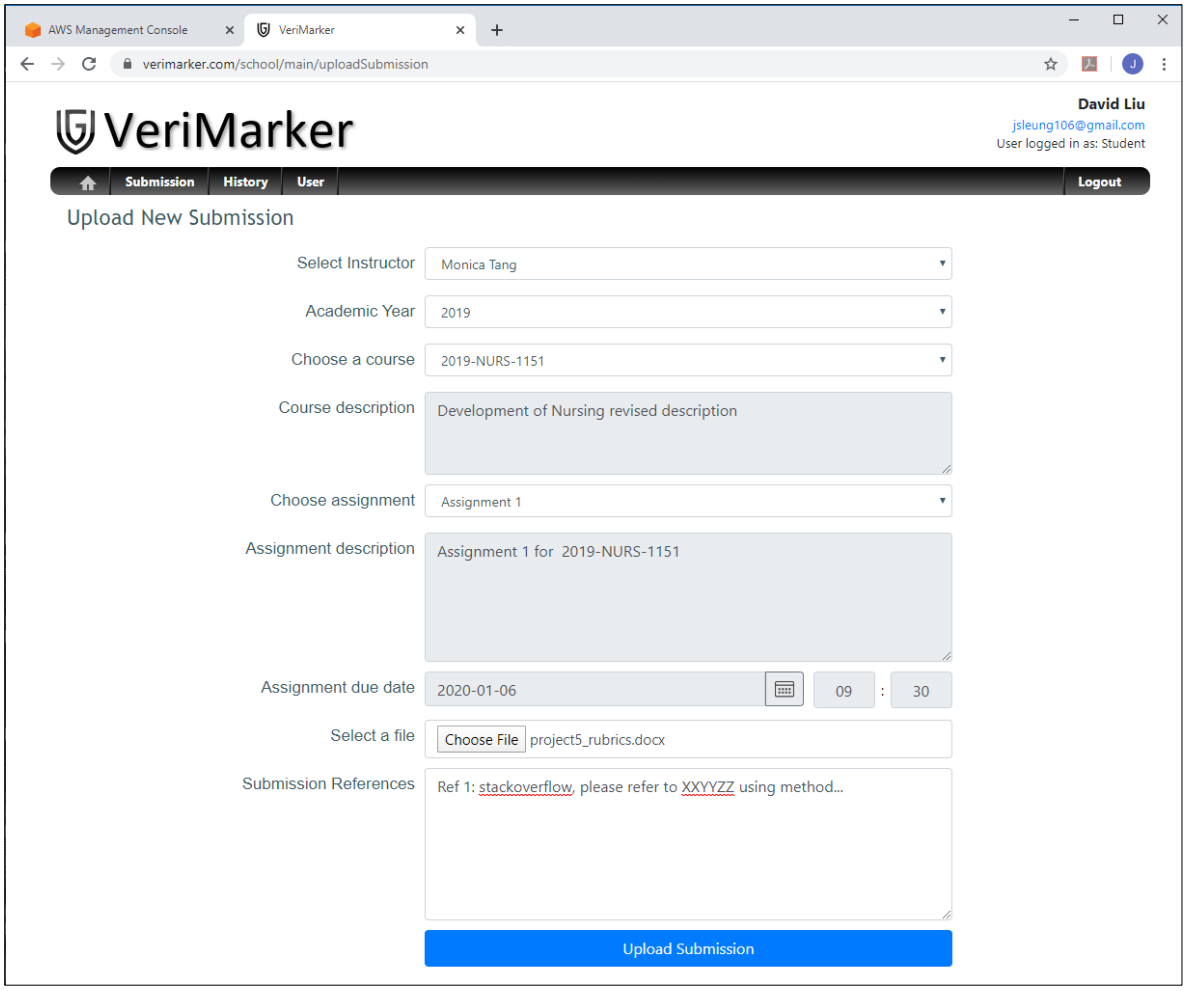
Section 3.10: Student "David Liu" to upload submission to Assignment 1 for Course "2019-NURS-1151" and check submission history

1. Login as David Liu. Initially, the submission history of David Liu should be empty:

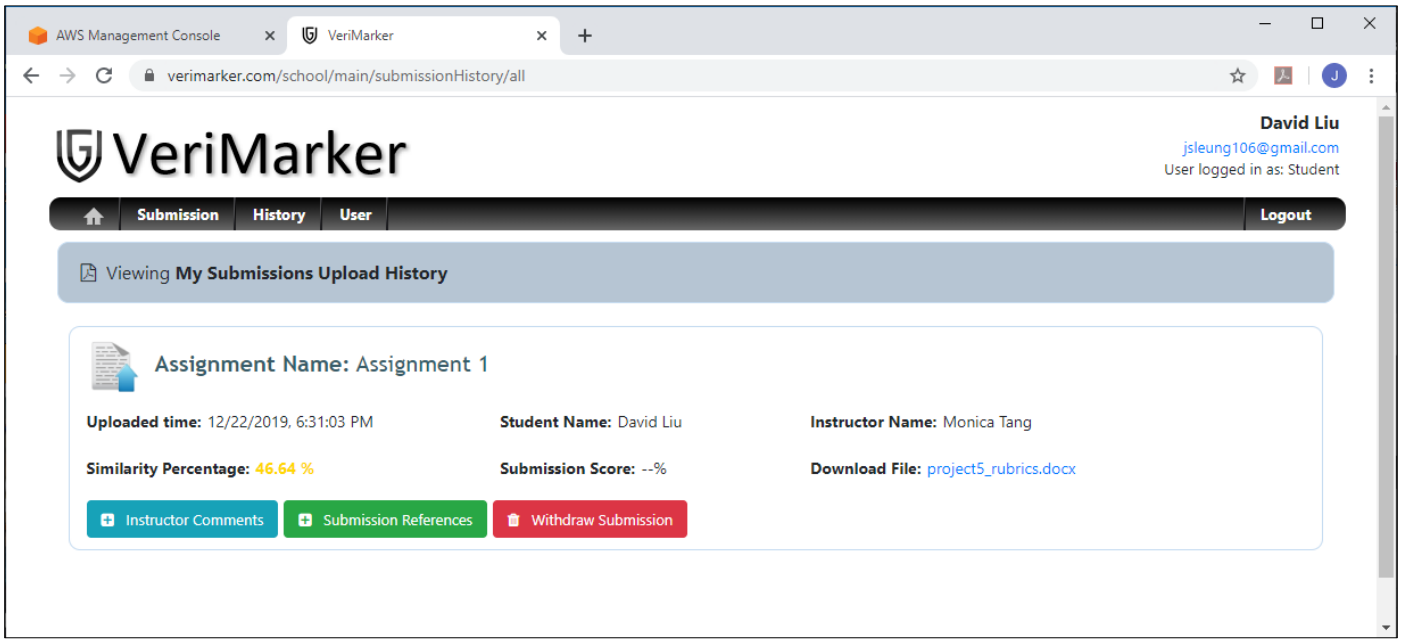


2. Click on Submission in the main menu. In the Upload New Submission page, select Instructor = Monica Tang, Academic Year = 2019, select course = 2019-NURS-1151, choose assignment = Assignment 1, select file = project5_rubrics.docx. Enter the appropriate Submission References and click Upload Submission:

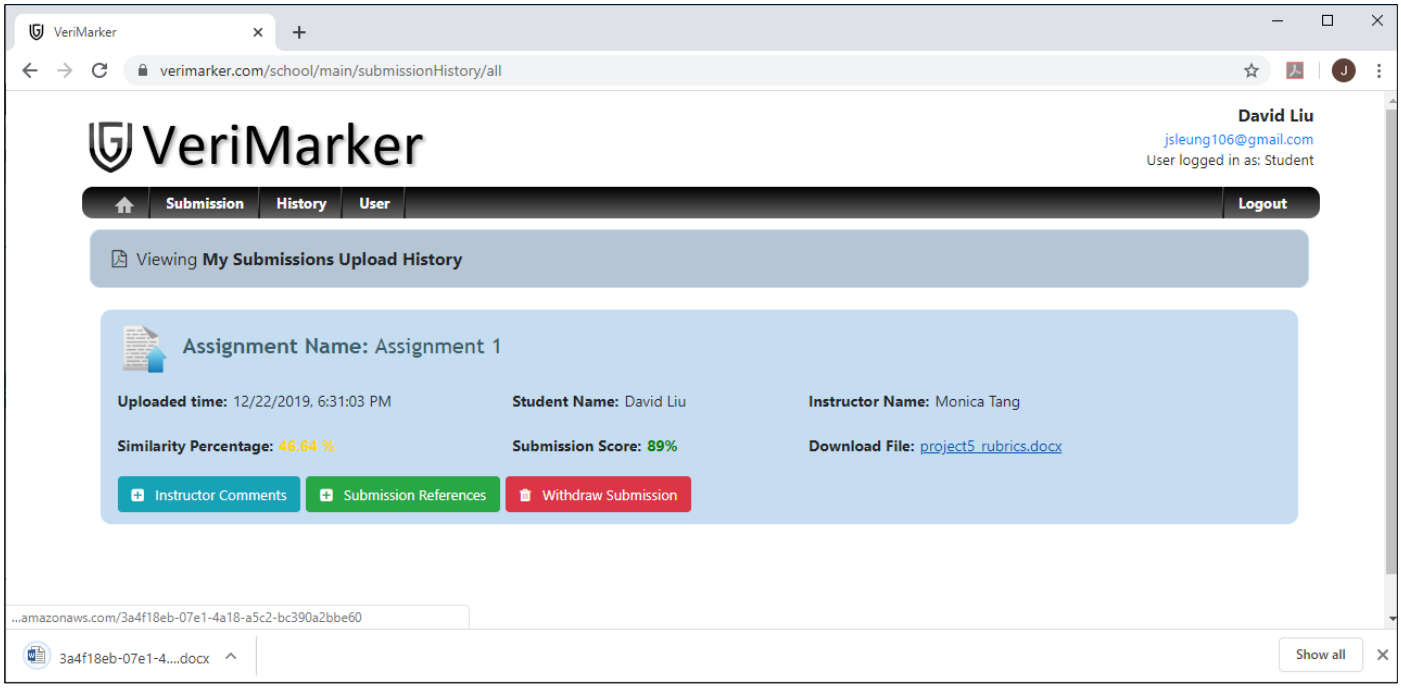
(continued to next page …)

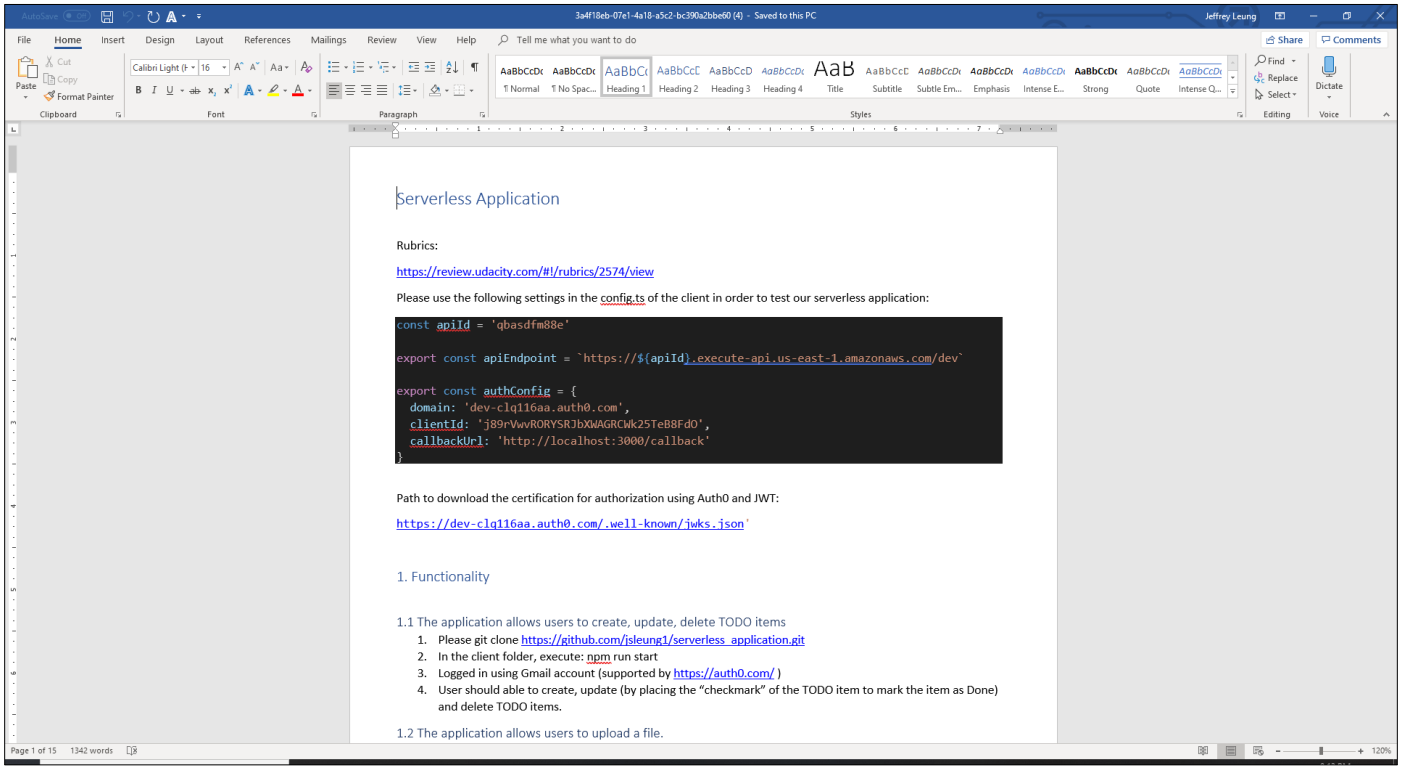


3. After the submission was successfully uploaded to VeriMarker, the submission history of David Liu will show the uploaded submission:



4. Click on the file name to download the file and verify the contents are correct:



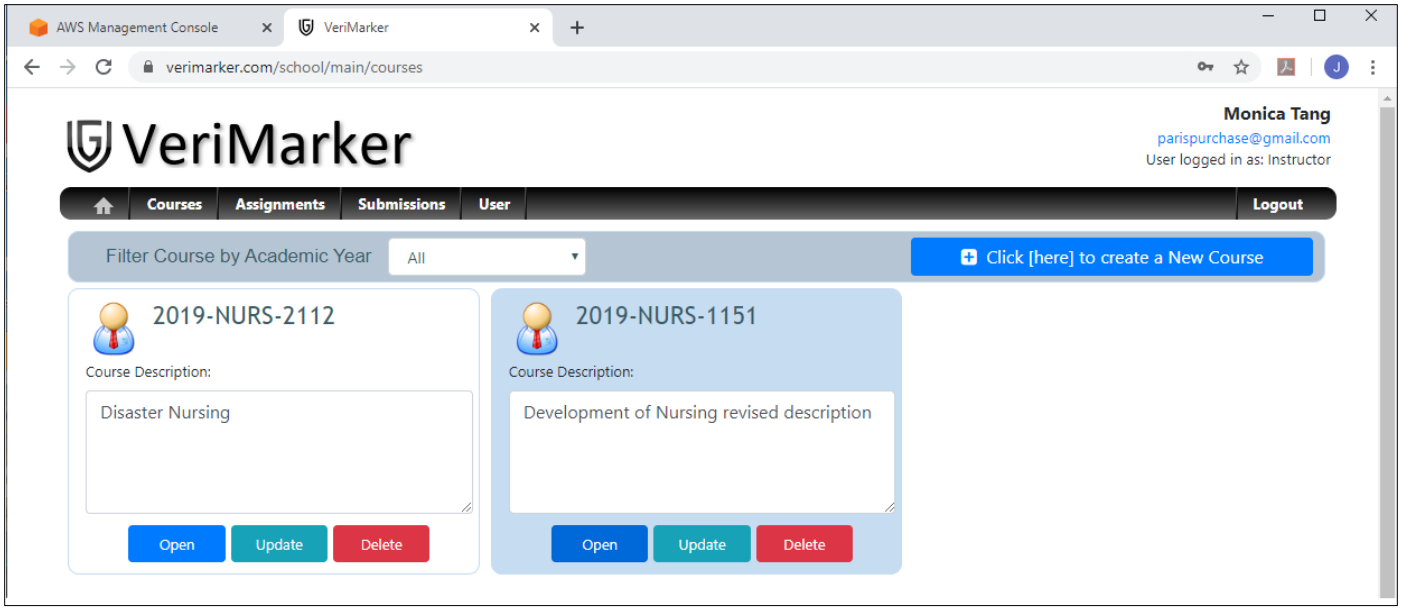


Section 3.11: Instructor to check on student submissions uploaded to assignment 1 in section 3.7 to 3.10.

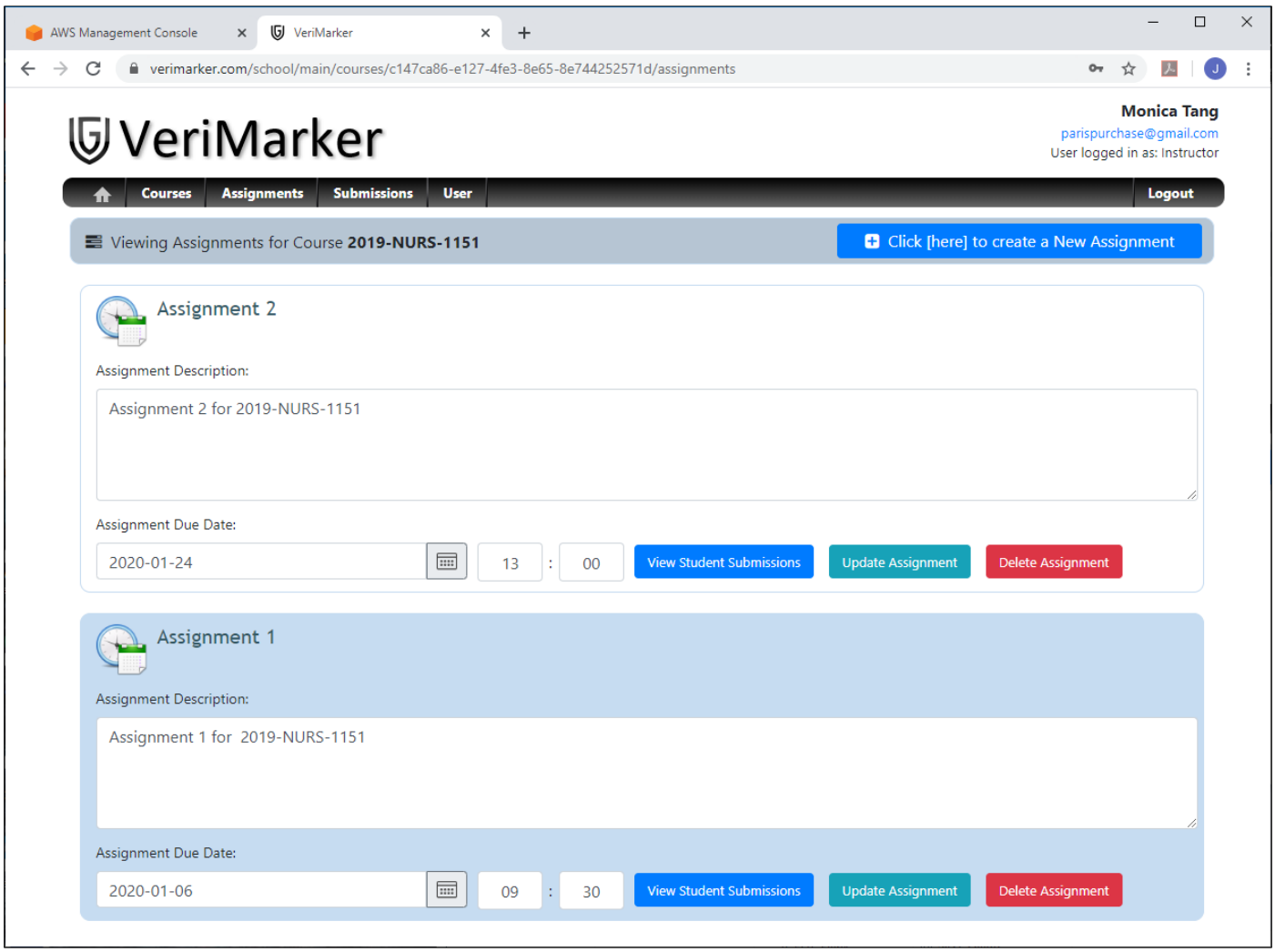
1. Login instructor "Monica Tang" (parispurchase@gmail.com) and Select My Courses in the main menu.

2. This will bring up the courses main page and display two courses that were created by Monica Tang. Select the course 2019-NURS-1151 and click Open:

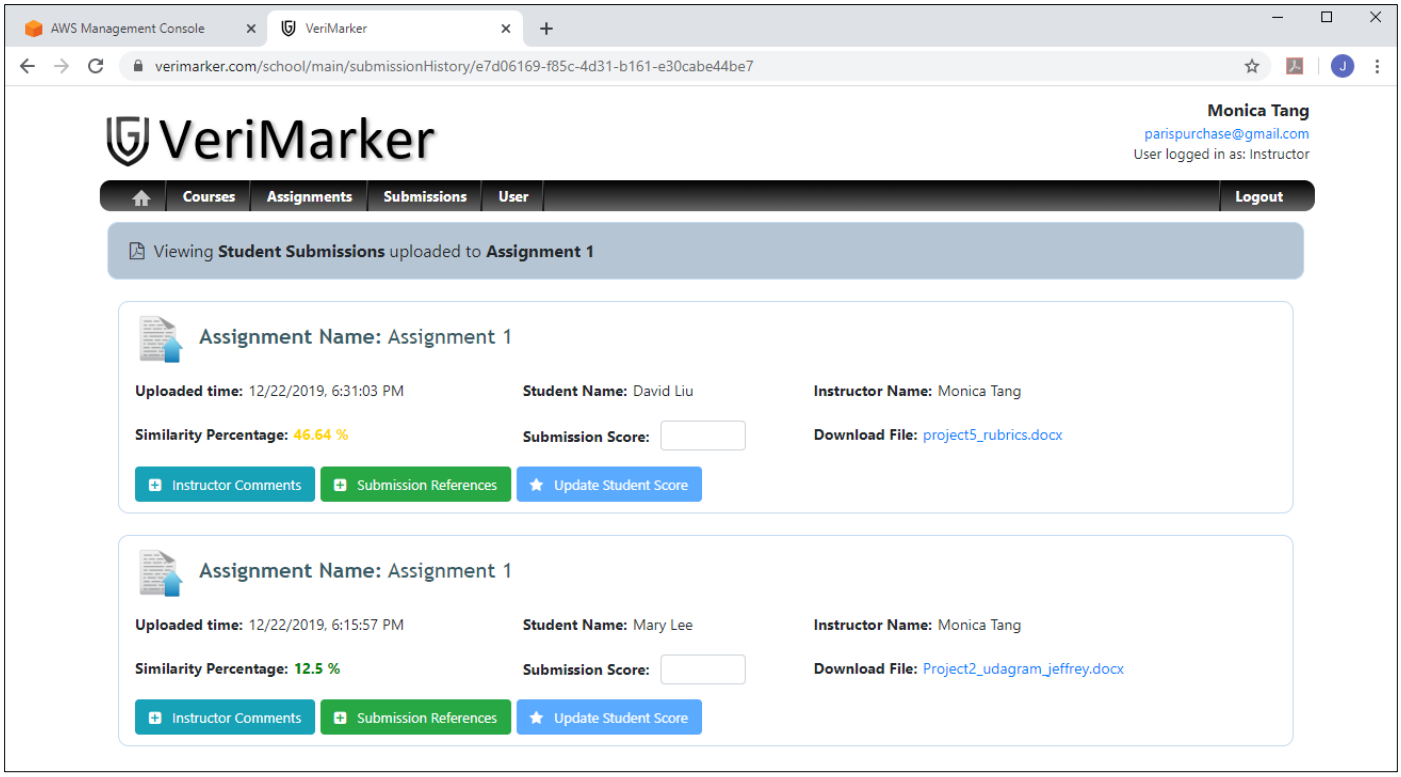
(continued to next page …)



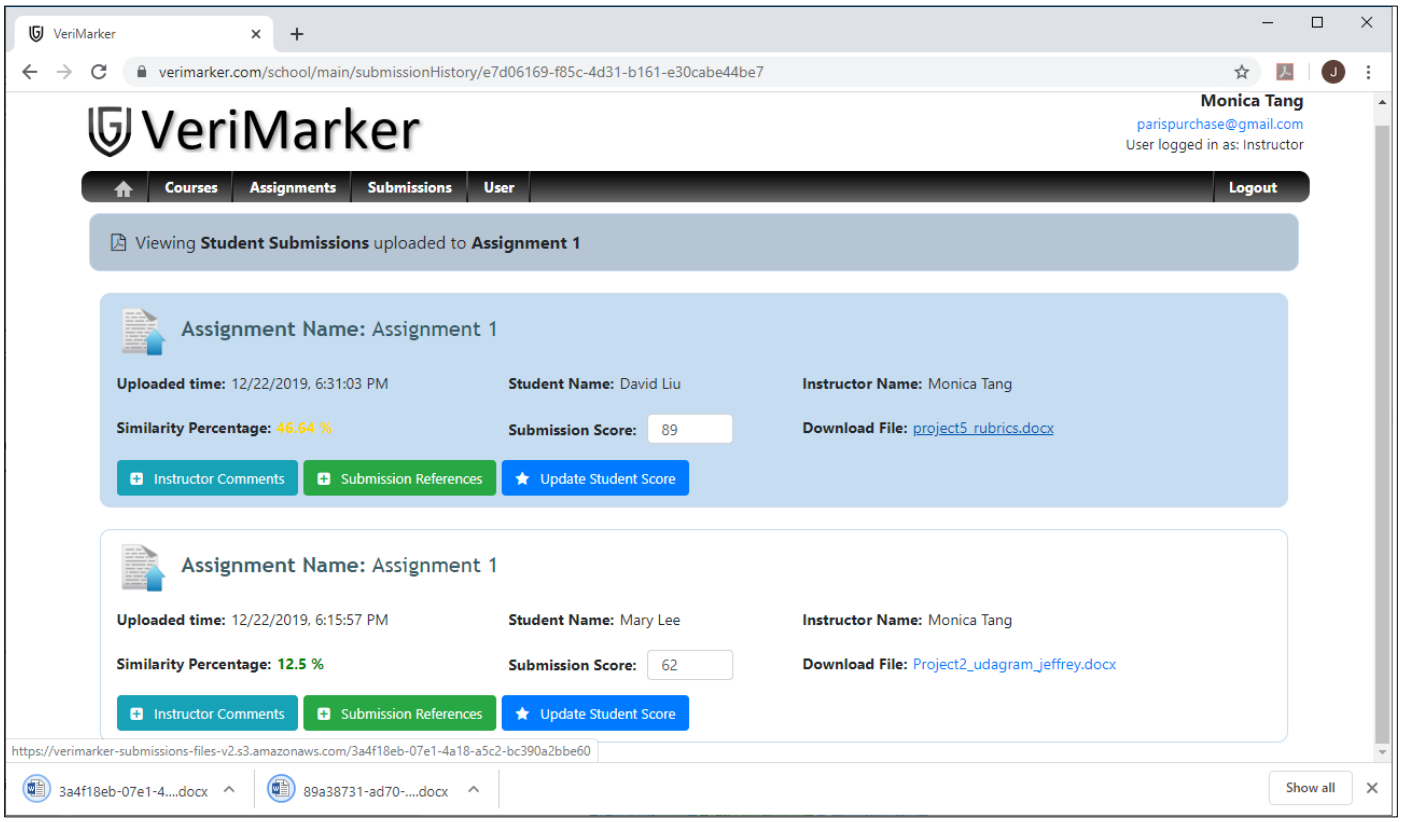
3. This will bring up the assignments page for Course 2019-NURS-1151. Select Assignment 1 and click View Student Submissions:



4. This will bring up the student submissions for Assignment 1 (in course 2019-NURS-1151). The following verifies that the submission of Mary Lee and David Liu were uploaded correctly to Instructor Monica Tang.

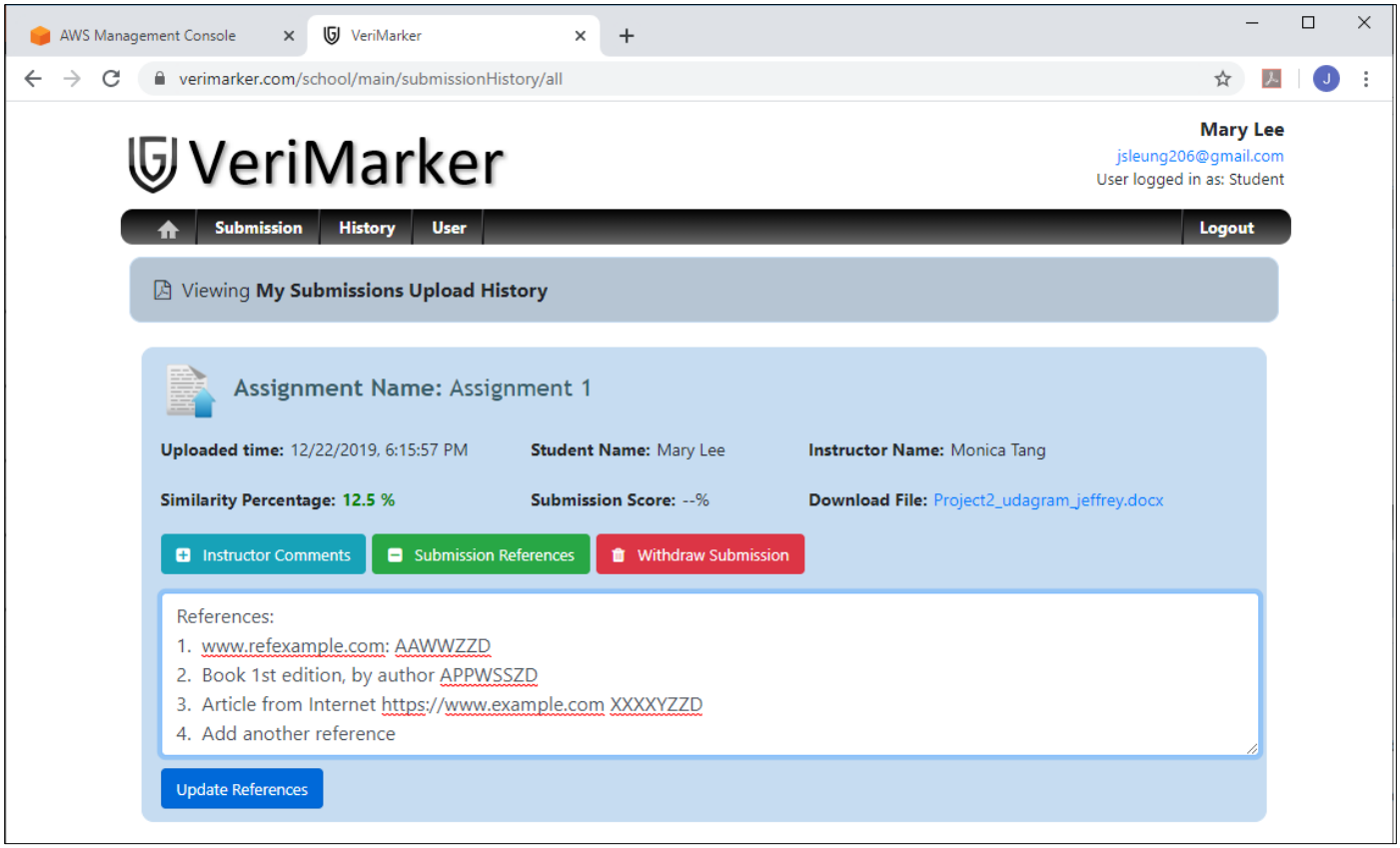


5. Click on the file name on both submissions should download the files uploaded by Mary Lee and David Liu. Verify the content of the files from both submissions are correct:

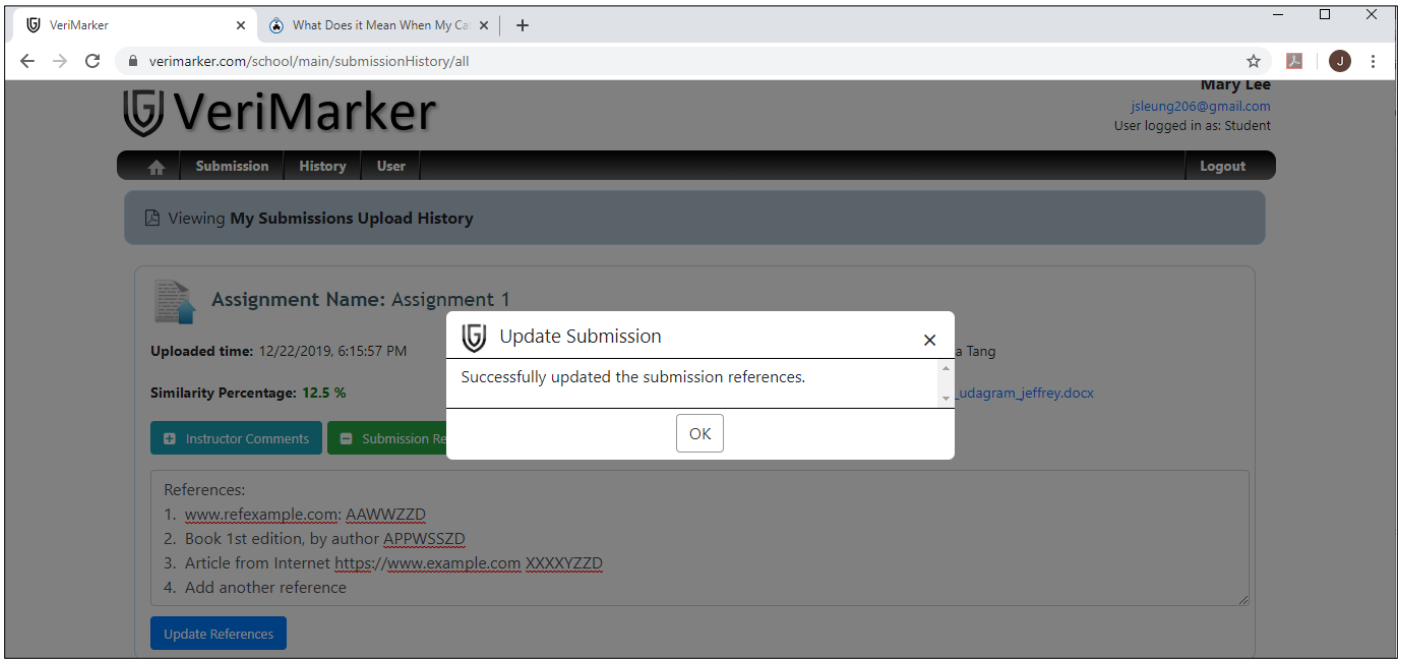


Section 3.12: Student update submission references

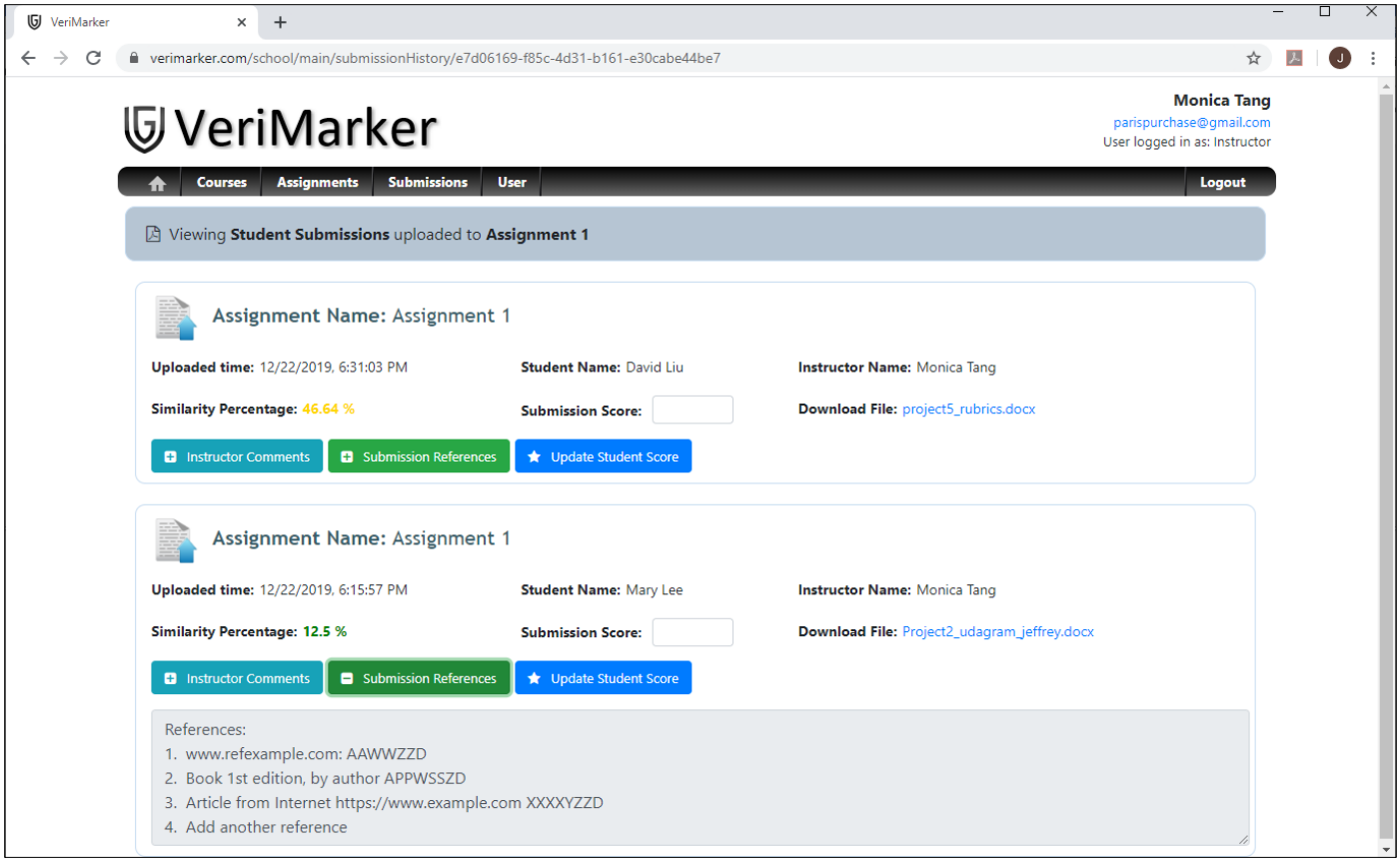
1. Login to VeriMarker using student Mary Lee (jsleung206@gmail.com). Click on Submission History in the main menu. Click on the "Submission References" button:



2. Add the last line ("4. Add another reference") and click on Update References button. The student's submission is updated with the latest References.



3. Login as Instructor Monica Tang, go to My Courses, select 2019-NURS-1151. Click on the View Student Submissions button of Assignment 1.



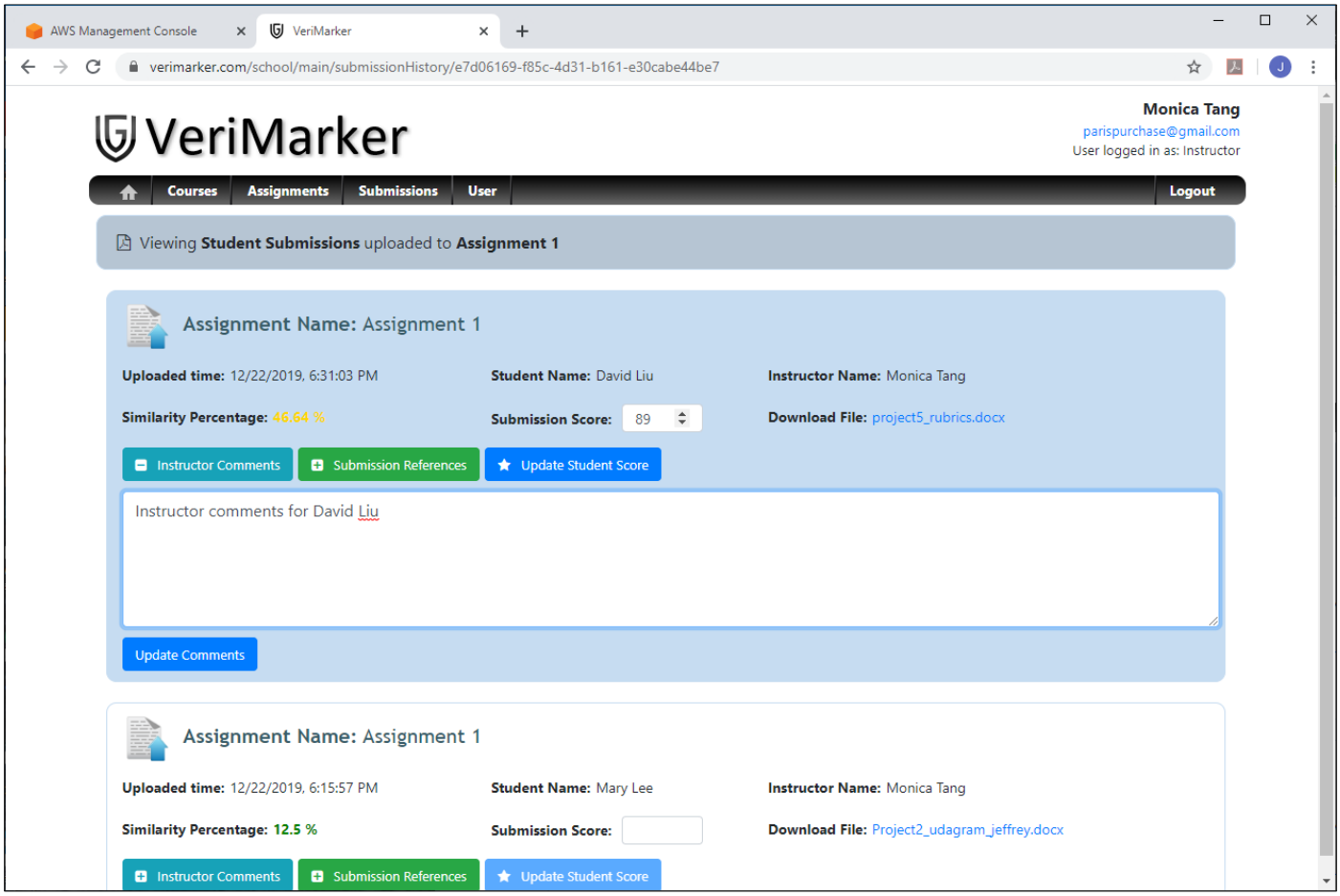
4. In the list of assignments, click on Submission References of Mary Lee. The instructor will be able to read the updated reference (last line as "4. Add another reference").

(continued to next page …)

Section 3.13: Instructor add comments and score to the student's submissions.

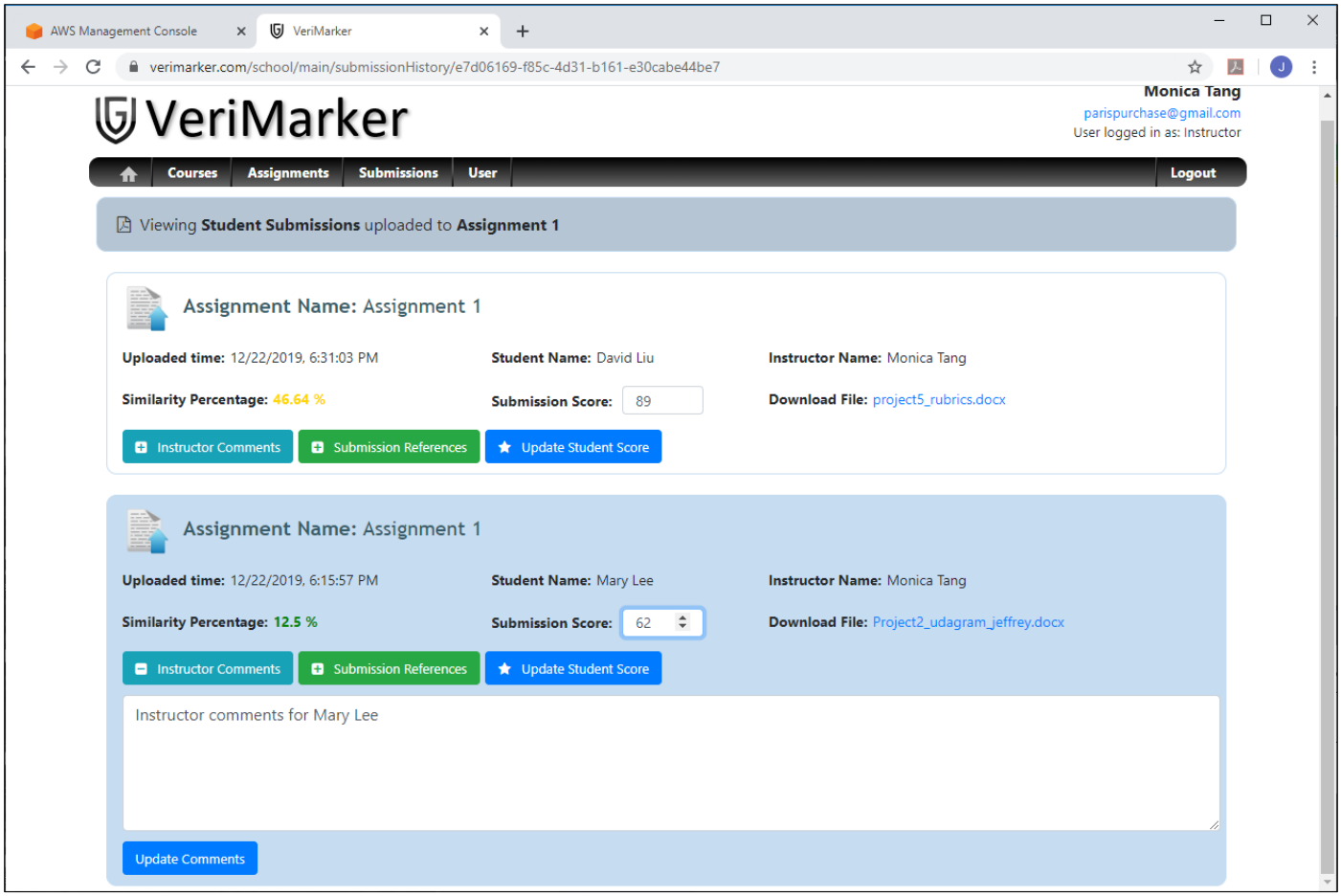
1. Continued from section 3.12, Instructor Monica Tang will add comments and assign score to the submissions of Mary Lee and David Liu.

2. To update David Liu's submission, instructor first assign 89 as the submission score and click on "Update Student Score". Then, Instructor click on the "Instructor Comments" button and add the comments "Instructor comments for David Liu". Click on the "Update Comments" button to update the instructor comments for the submission:



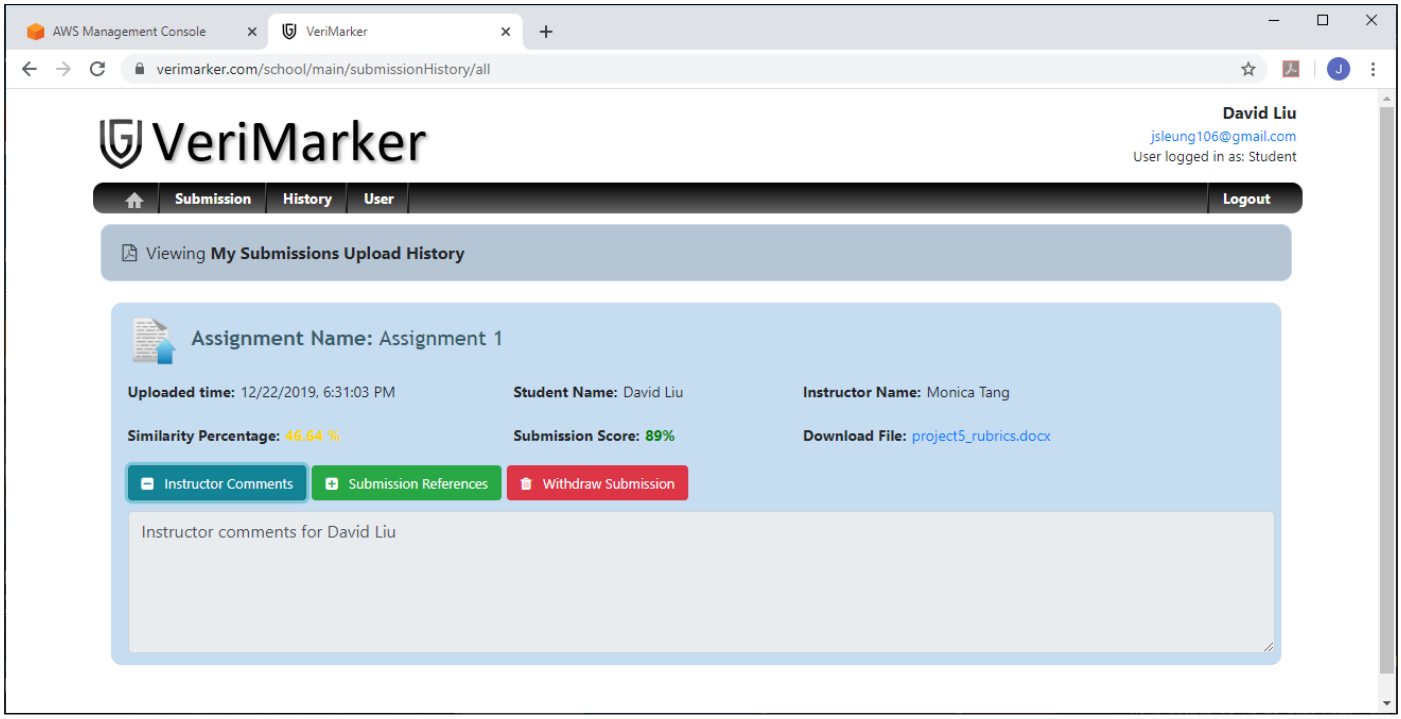
(continued to next page …)

3. Similarly, to update Mary Lee's submission, instructor first assign 62 as the submission score and click on "Update Student Score". Then, Instructor click on the "Instructor Comments" button and add the comments "Instructor comments for Mary Lee". Click on the "Update Comments" button to update the instructor comments for the submission:

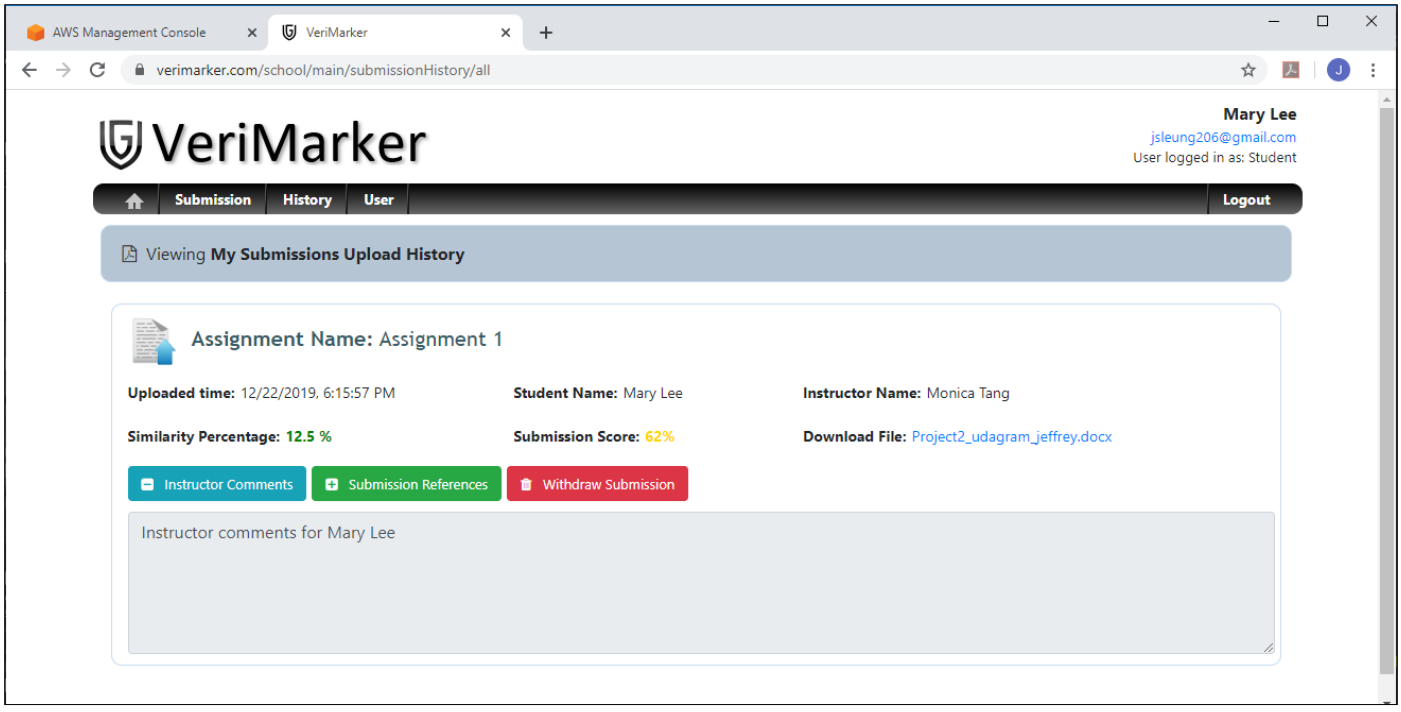


4. Logged in as student David Liu. In the submission history of David Liu, the only one submission in Submission History should display submission score 89 and instructor comments "Instructor comments for David Liu":

(continued to next page …)



5. Similarly, logged in as student Mary Lee. In the submission history of Mary Lee, the only one submission in Submission History should display submission score 62 and instructor comments "Instructor comments for Mary Lee":



4. Project Rubrics

Project Setup:

Refer to **Option 2** of the Cloud Rubrics for the Capstone Project:
https://review.udacity.com/#!/rubrics/2578/view

The project structure of VeriMarker is very similar to the one in Serverless project:

1. “backend” folder – contains serverless and lambda function implementations.

2. “client” folder – contains the code for the Angular front-end.

One particular file is the config.ts (located in client/src/app/config.ts) in the client project. This file contains the parent URL request path to the AWS Lambda functions, and the Auth0 settings such as clientId and callbackUrl. Please note Auth0 settings (authConfig_prod) is commented out, since starting the local client will use a different set of Auth0 settings (authConfig_dev) as indicated in the following:

```
const apiId = 'lh756iw3bi';
const env = 'v2';

export const apiEndpoint = `https://${apiId}.execute-api.us-east-1.amazonaws.com/${env}`
/*
export const authConfig_prod = {
  domain: 'dev-clq116aa.auth0.com',
  clientId: 'QLaMP23u6l9whbZWmXf6PTF9eKxMSVU8',
  callbackUrl: 'https://www.verimarker.com/school/auth0',
  mode: 'prod'
}
*/
export const authConfig_dev = {
  domain: 'dev-clq116aa.auth0.com',
  clientId: 'Z52r8N5nV8h3XvWr3jqZJm9wE3JWi6yq',
  callbackUrl: 'http://localhost:4200/school/auth0',
  mode: 'dev'
}

export const authConfig = authConfig_dev
```

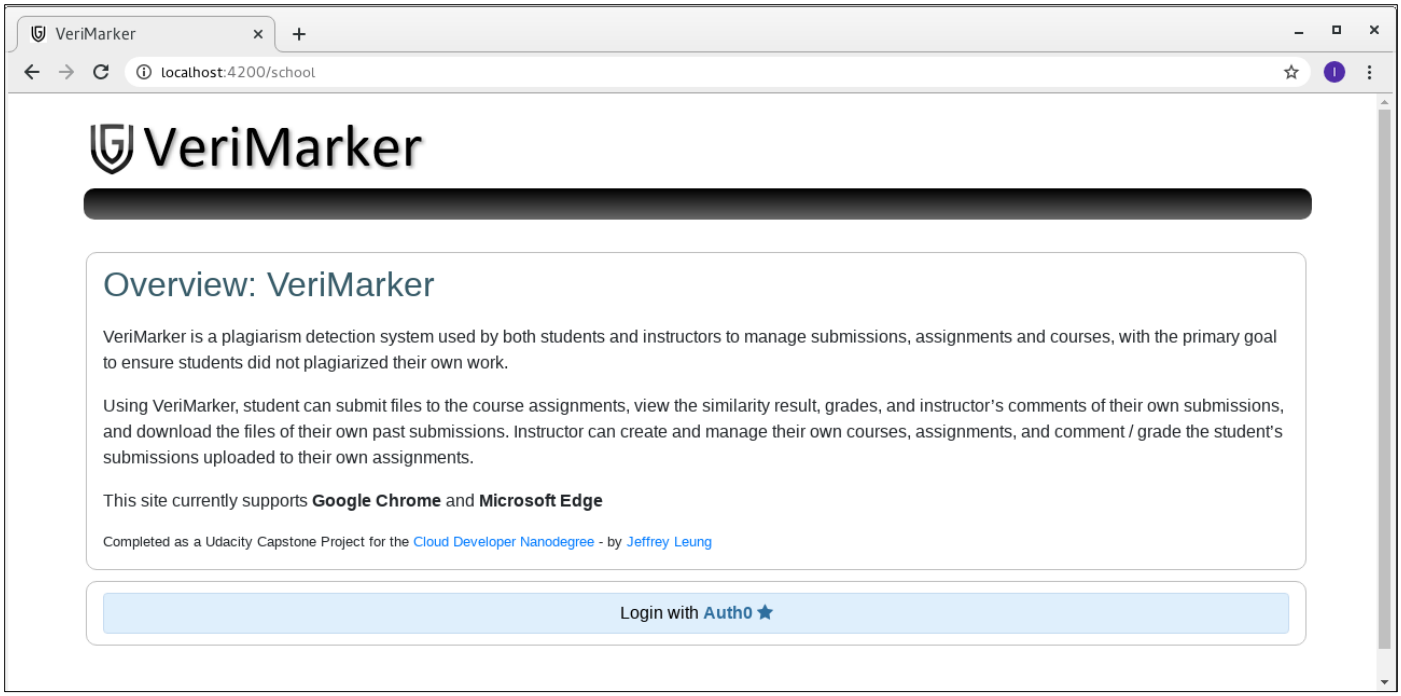
Path to download the certification for authorization using Auth0 and JWT:
https://dev-clq116aa.auth0.com/.well-known/jwks.json'

The Postman REST API for VeriMarker can be imported from the file: capstone_project.postman_collection.json (located in the project root directory).

To run the Angular client, please perform the following steps:

1. git clone https://github.com/jsleung1/project_capstone_verimarker.git
2. In Visual Studio Code, open the project folder “project_capstone_verimarker”.
3. Open a terminal in Visual Studio code, and cd to the client folder (project_capstone_verimarker/client).
4. In the client folder, install latest Angular by execute: npm install -g @angular/cli (you can try install Angular locally to the project by: npm install @angular/cli , but you need additional setup for this to work).

5. Verify Angular install correctly by Execute: ng –version (it should return Angular 8.0.6).
6. Start the Angular 8, execute: npm run start
7. In Chrome Browser, go to the Url: http://localhost:4200
8. Please use your Gmail account to login VeriMarker:



4.1 Functionality

4.1.1 The application allows users to create, update, delete items

Click on the link in the following table to go through the details of the Unit Test that shows how VeriMarker allows both Instructors and Students to create, update and delete items:

Section	Unit Test Name	User Type
3.2	Instructor create course / update course description	Instructor
3.3	Instructor delete a course	Instructor
3.4	Instructor create assignments	Instructor
3.5	Instructor delete assignment	Instructor
3.9	Student “Mary Lee” deleted the second submission uploaded in previous section (3.8)	Student
3.12	Student update the submission references (submission update)	Student
3.13	Instructor add comments and score to the student’s submissions (submission update)	Instructor

4.1.2 The application allows users to upload a file.

Click on the link in the following table to go through the details of the Unit Test that shows how VeriMarker allows Students to upload files and download the file with the correct content:

Section	Unit Test Name	User Type
3.7	Student “Mary Lee” upload submission to the course assignment and check her submission history.	Student
3.8	Student “Mary Lee” upload the second submission to the same course and assignment.	Student
3.10	Student “David Liu” to upload submission to Assignment 1 for Course “2019-NURS-1151”.	Student
3.11	Instructor to check on student submissions uploaded to assignment 1 in section 3.7 to 3.10.	Instructor

4.1.3 The application only displays items for a logged in user.

Click on the link in the following table to go through the details of the Unit Test that shows how VeriMarker only display instructor own courses and assignments, and student only view their own submissions:

Section	Unit Test Name	User Type
3.2	Instructor create course / update course description (only display list of courses created by the instructor)	Instructor
3.4	Instructor create assignments (only display list of assignments created by the instructor)	Instructor
3.7	Student “Mary Lee” upload submission to the course assignment and check her submission history (submission history only showed her own uploaded submission).	Student
3.8	Student “Mary Lee” upload the second submission to the same course and assignment (submission history only showed her own uploaded submissions).	Student
3.10	Student “David Liu” to upload submission to Assignment 1 for Course “2019-NURS-1151” and check submission history (submission history only showed his own uploaded submission).	Student

Please also refer to one of the data access layers (located in backend/src/dataLayer) to query user items.

In the follow example, when the student clicks on Submission History, submissionAccess.ts performs a query of Submission items using the student Id (which is the value of the decodedJwt.sub in the JwtToken):

```
    async getAllSubmissionsByStudentId(studentId: string): Promise<Submission[]> {

        const result = await this.docClient.query({
          TableName: this.submissionsTable,
          IndexName: this.submissionsStudentIdIndex,
          KeyConditionExpression: 'studentId = :studentId',
          ExpressionAttributeValues: {
              ':studentId': studentId
          },
```

```
          ScanIndexForward: false
        }).promise()

        const items = result.Items
        return items as Submission[]
    }
```

Another example is when the Instructor clicks on “My Courses” in the main menu. The courseAccess.ts performs a query of Courses items using the instructor Id (which is the value of the decodedJwt.sub in the JwtToken):

```
    async getAllCoursesByInstructorId(instructorId: string): Promise<Course[]> {

        const result = await this.docClient.query({
          TableName: this.coursesTable,
          IndexName: this.coursesInstructorIdIndex,
          KeyConditionExpression: 'instructorId = :instructorId',
          ExpressionAttributeValues: {
              ':instructorId': instructorId
          },
          ScanIndexForward: false
        }).promise()

        const items = result.Items
        return items as Course[]
    }
```

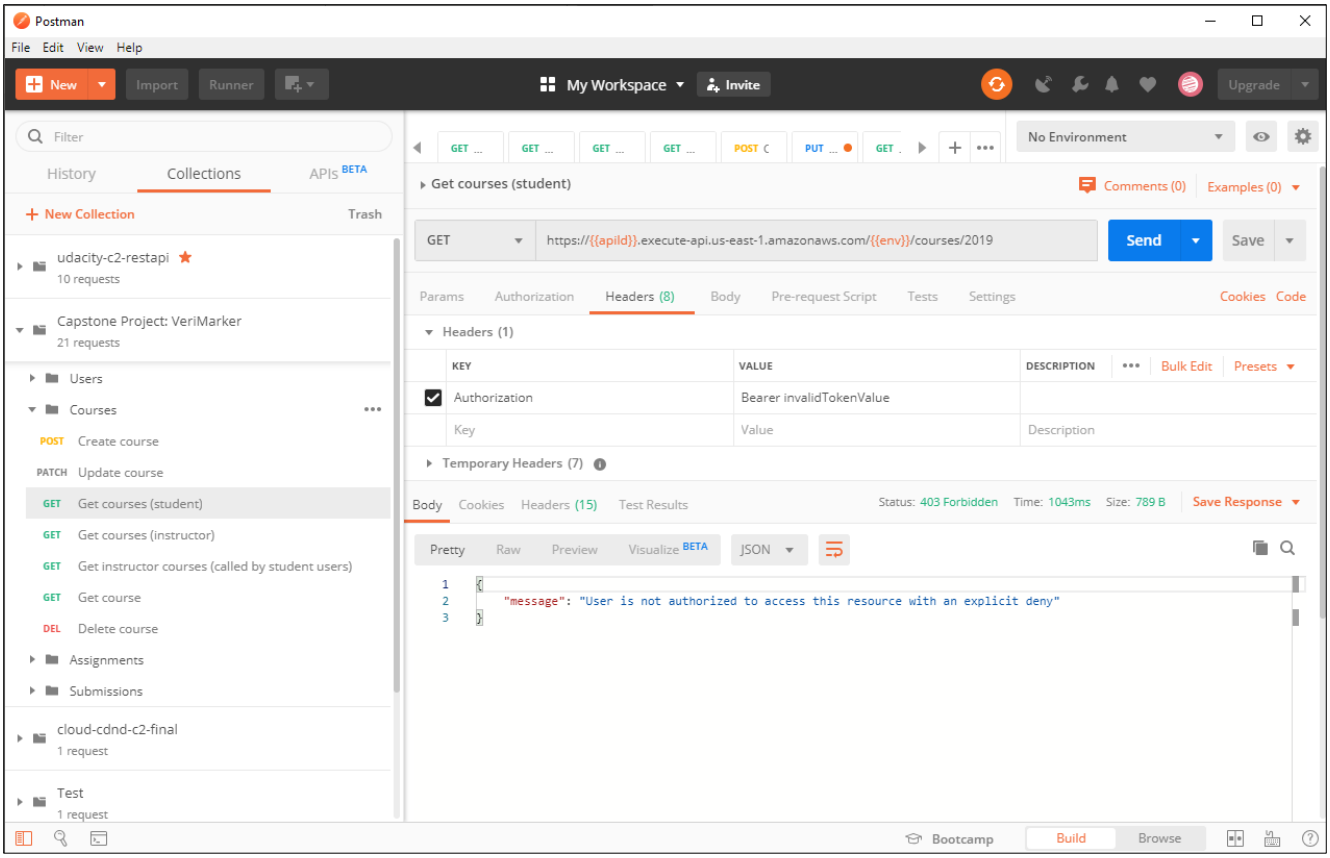
4.1.4 Authentication is implemented and does not allow unauthenticated access.

All of the AWS Lambda functions defined in serverless.yml are declared with the authorizer Auth0, which is the auth0Authorizer.ts (located in backend/src/lambda/auth). The bulk of the work in auth0Authorizer.ts is call the verifyToken function:

```
async function verifyToken(authHeader: string): Promise<JwtPayload> {
  const token = getToken(authHeader)
  const cert = await getJksCert( jwksUrl )
  return verify(token, cert, { algorithms: ['RS256'] }) as JwtPayload
}
```

First, it gets the token from the authorization header. Then, it calls async method getJksCert with url path https://dev-clq116aa.auth0.com/.well-known/jwks.json . This will download the certificate to verify the incoming token from the request header, such as whether the token already expired, or simply an invalid token.

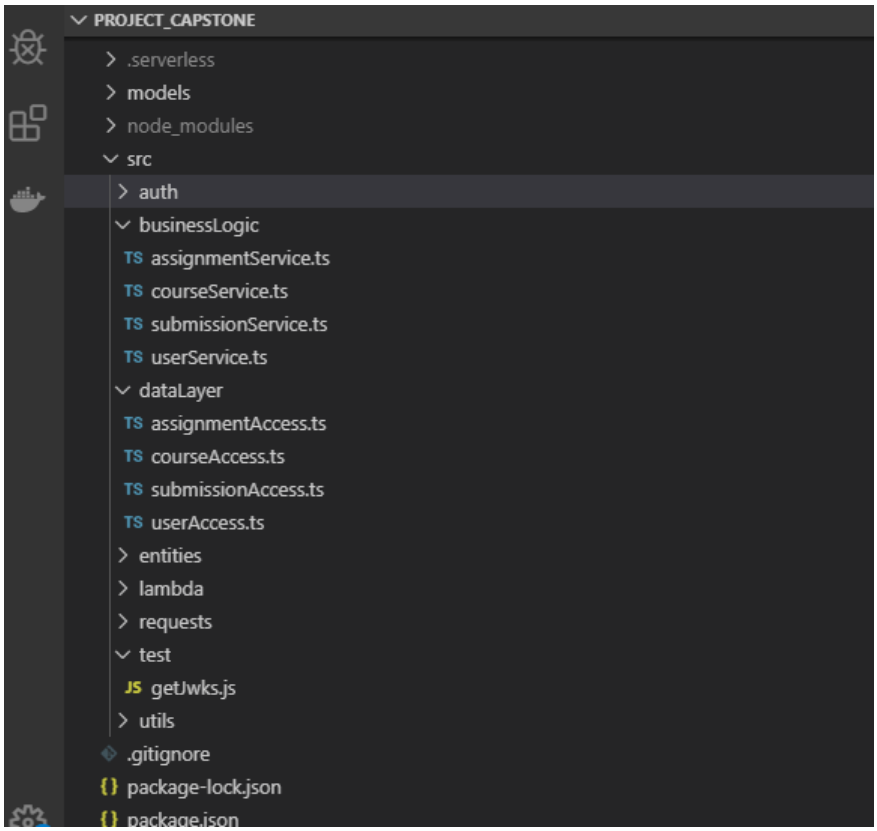
To try it out, in Postman, using the wrong jwtToken value or missing Authorization header will result in a HTTP respond with error code 403 Forbidden as shown in the following example:



4.2. Code Base

4.2.1 The code is split into multiple layers separating business logic from I/O related code.

Refer to the code structure in the /backend folder, the code is split into businessLogic and dataLayer (I/O) layers:



The following table summarized the functions of the business logic and data layer (I/O) in VeriMarker:

Business Logic (src/businessLogic)	**Description**
courseService.ts	- Check the userType. If the user is an Instructor, get the courses created by that instructor for that academic year. If the user is a Student, get courses of ALL instructors for that academic year. - Check the userType is an Instructor before allowing to create, update and delete Course. Also, check if the instructor is the course owner before allowing to update and delete the course. - Check to ensure no assignments are created under the course before the Instructor can delete the course.
assignmentService.ts	- Get all the assignments of the course for the student. For the instructor, check if the instructor is the course creator before return all the assignments of the course. - Check the userType is an Instructor before allowing to create, update or delete the assignment. Check to ensure no submissions are uploaded to the assignment before the Instructor can delete the assignment.
submissionService.ts	- Get all submissions uploaded by the student. For the instructor, return only the submissions uploaded to the Instructor by the student. - Check the user is a student before allowing to delete the submission, or update the submission references. - Check the user is an instructor before allowing to update the submission score and instructor's comments.
userService.ts	- Check if the user is registered in VeriMarker by using the userId which is the value of the decodedJwt.sub in the JwtToken. If the user is not registered in VeriMarker, returns an empty JSON user to the Angular client. - Create a new registered user in VeriMarker. - Allows user to update the email and user Type (Student or Instructor).

DataLayer (I/O) (src/dataLayer)	**Description**
courseAccess.ts assignmentAccess.ts submissionAccess.ts userAccess.ts	- Enable AWS Ray X-Ray. - Performs creation of item in DynamoDb using docClient.put - Performs query of DynamoDb using docClient.query, using the appropriate Indexes. - Performs update of item in DynamoDb using docClient.update, using the appropriate UpdateExpression and ExpressionAttributeValues. - Performs delete of Course item in DynamoDb using docClient.delete - All queried items are sorted by returning the most recent created item first (using ScanIndexForward: false on the range key "CreatedAt").

4.2.2 Code is implemented using async/await and Promises without using callbacks.

Please check all the code in the backend folder. It contains code using async/await and Promises without using callbacks.

4.3. Best Practices

4.3.1 All resources in the application are defined in the "serverless.yml" file

All resources are defined in backend/serverless.yml . Please check it out.

4.3.2 Each function has its own set of permissions.

In the serverless.yml file, under plugins, the “serverless-iam-roles-per-function” was included. Please check the serverless.yml which shows all the Lambda functions has its own iamRoleStatements. The following is a snippet of the serverless.yml, which shows the iamRoleStatements for the GetUserSubmissions Lambda function:

```
  GetUserSubmissions:
    handler: src/lambda/http/submission/getUserSubmissions.handler
    events:
      - http:
          method: get
          path: submissions
          cors: true
          authorizer: Auth0
    iamRoleStatements:
    - Effect: Allow
      Action:
        - dynamodb:Query
      Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environme
nt.USERS_TABLE}
    - Effect: Allow
      Action:
        - dynamodb:Query
      Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environme
nt.USERS_TABLE}/index/${self:provider.environment.USERS_USERID_INDEX}
    - Effect: Allow
      Action:
        - dynamodb:Query
      Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environme
nt.SUBMISSIONS_TABLE}/index/${self:provider.environment.SUBMISSIONS_STUDENTID_INDEX}
    - Effect: Allow
      Action:
        - dynamodb:Query
      Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environme
nt.SUBMISSIONS_TABLE}/index/${self:provider.environment.SUBMISSIONS_INSTRUCTORID_INDEX}

    - Effect: Allow
      Action:
        - dynamodb:Query
```

```
      Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environment.SUBMISSIONS_TABLE}
    - Effect: Allow
      Action:
        - xray:PutTraceSegments
        - xray:PutTelemetryRecords
      Resource: "*"
```

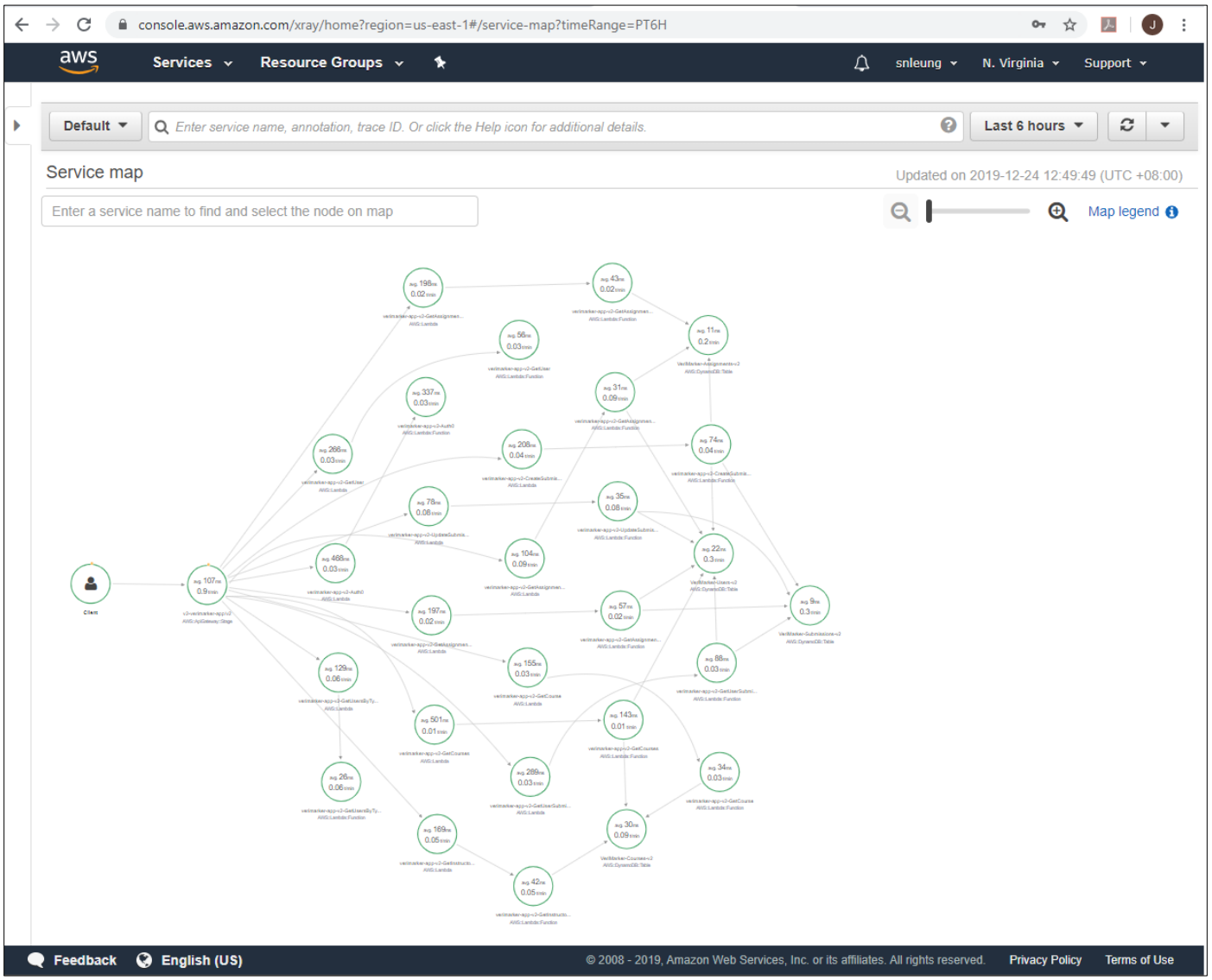
4.3.3 Application has sufficient monitoring.

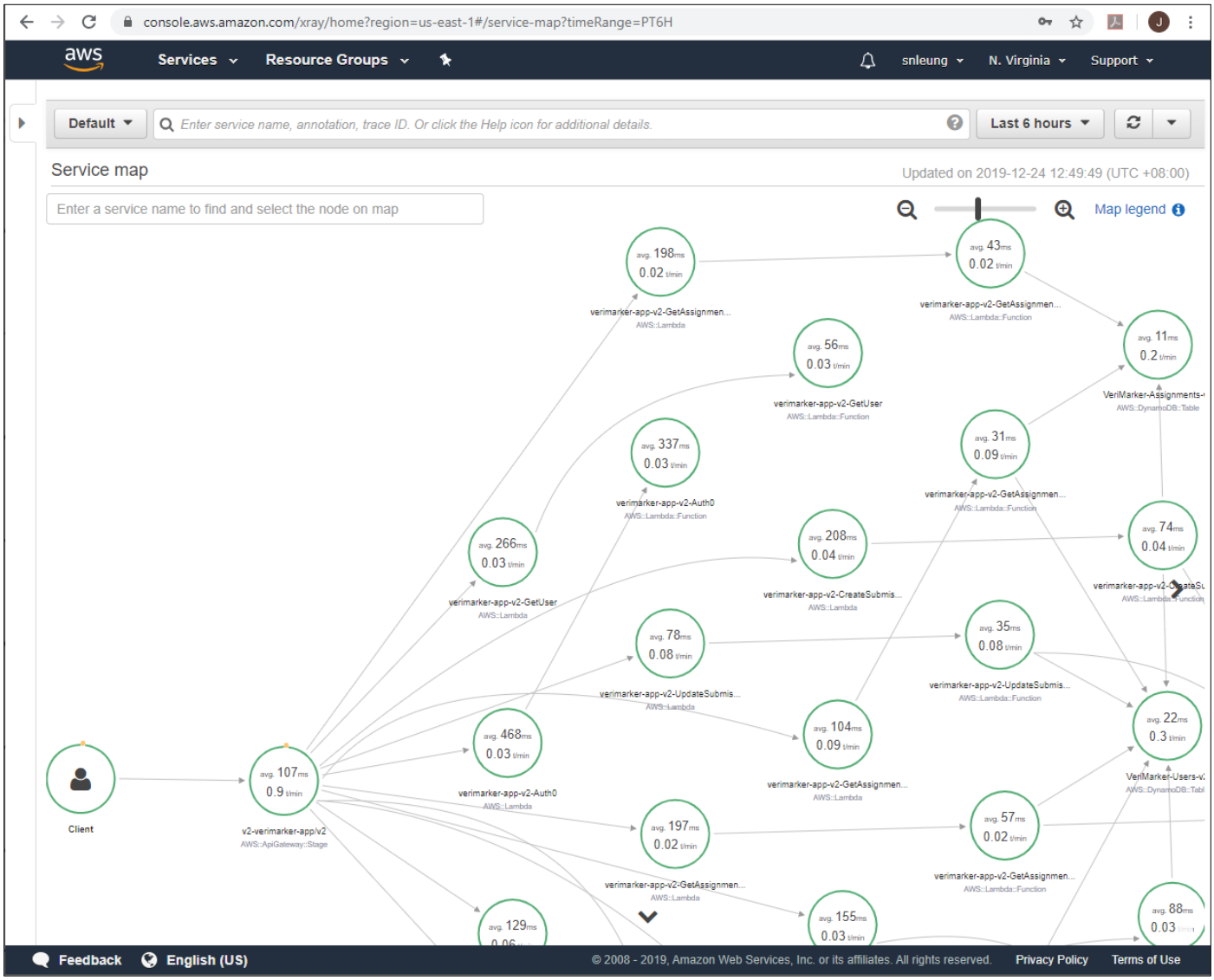
- Distributed tracing is enabled

Using AWS X-Ray, distributed tracing is enabled by define the following in serverless.yml:

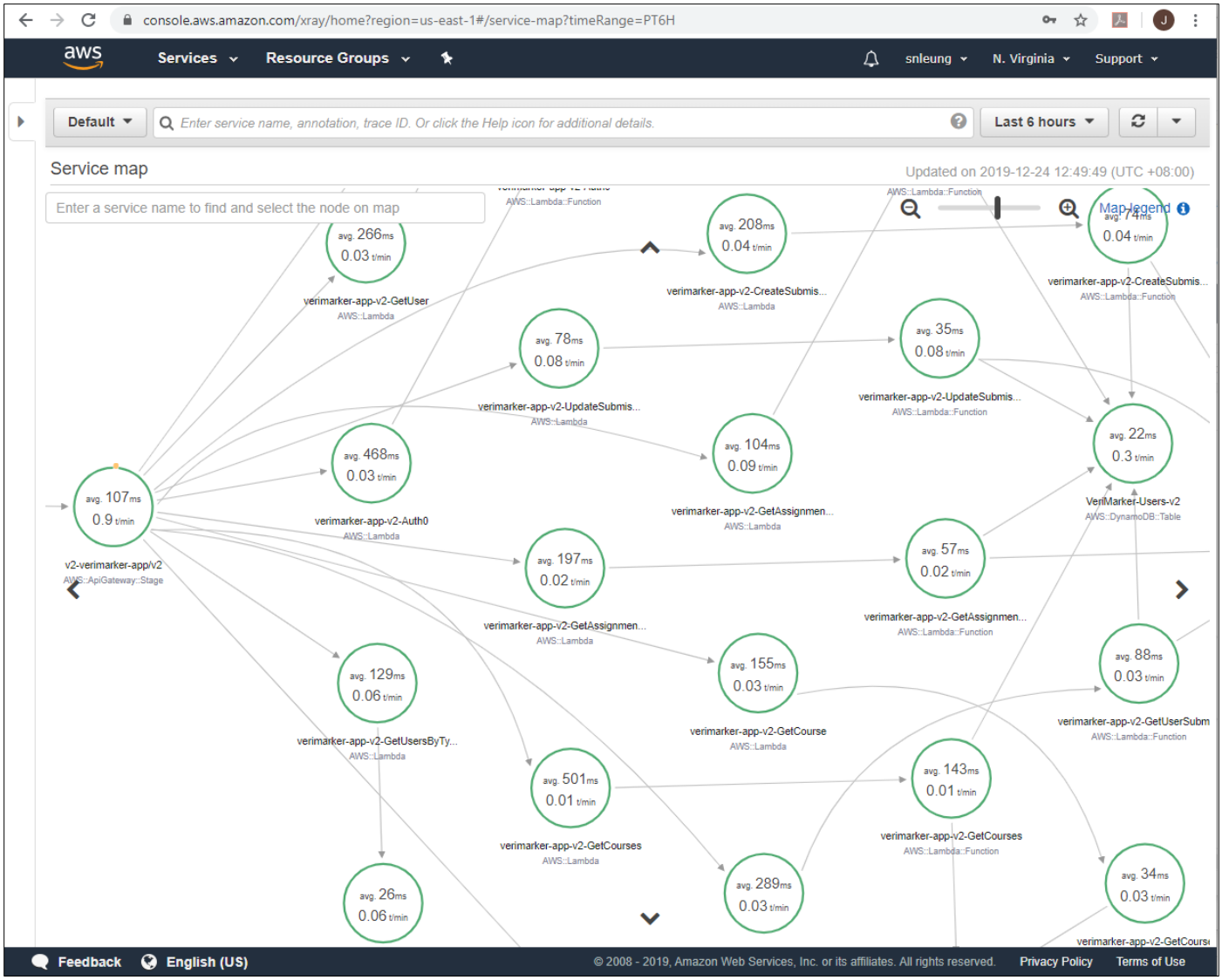
```
provider:
  ....
  tracing:
    lambda: true
    apiGateway: true
```

In the AWS Console, under AWS X-Ray, it shows the following trace in the following diagrams:





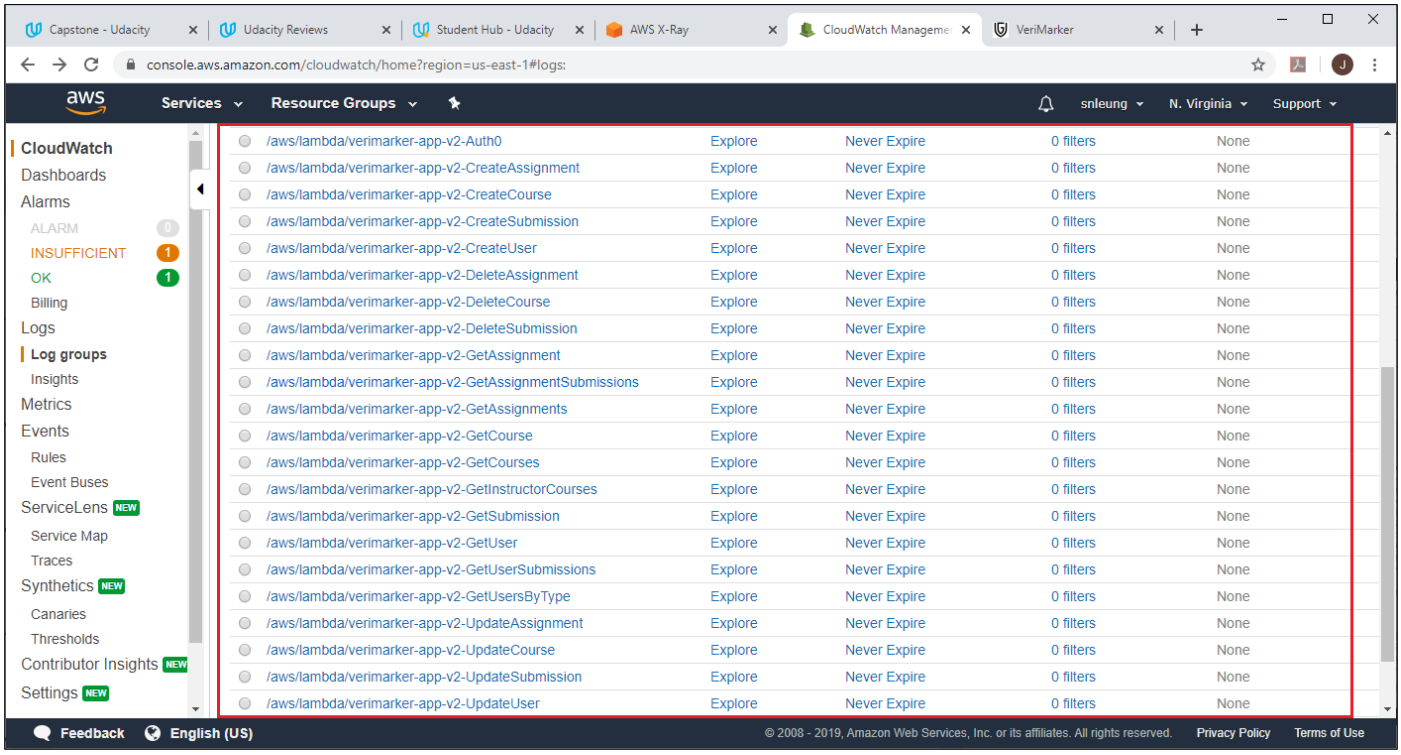
(continued to next page …)



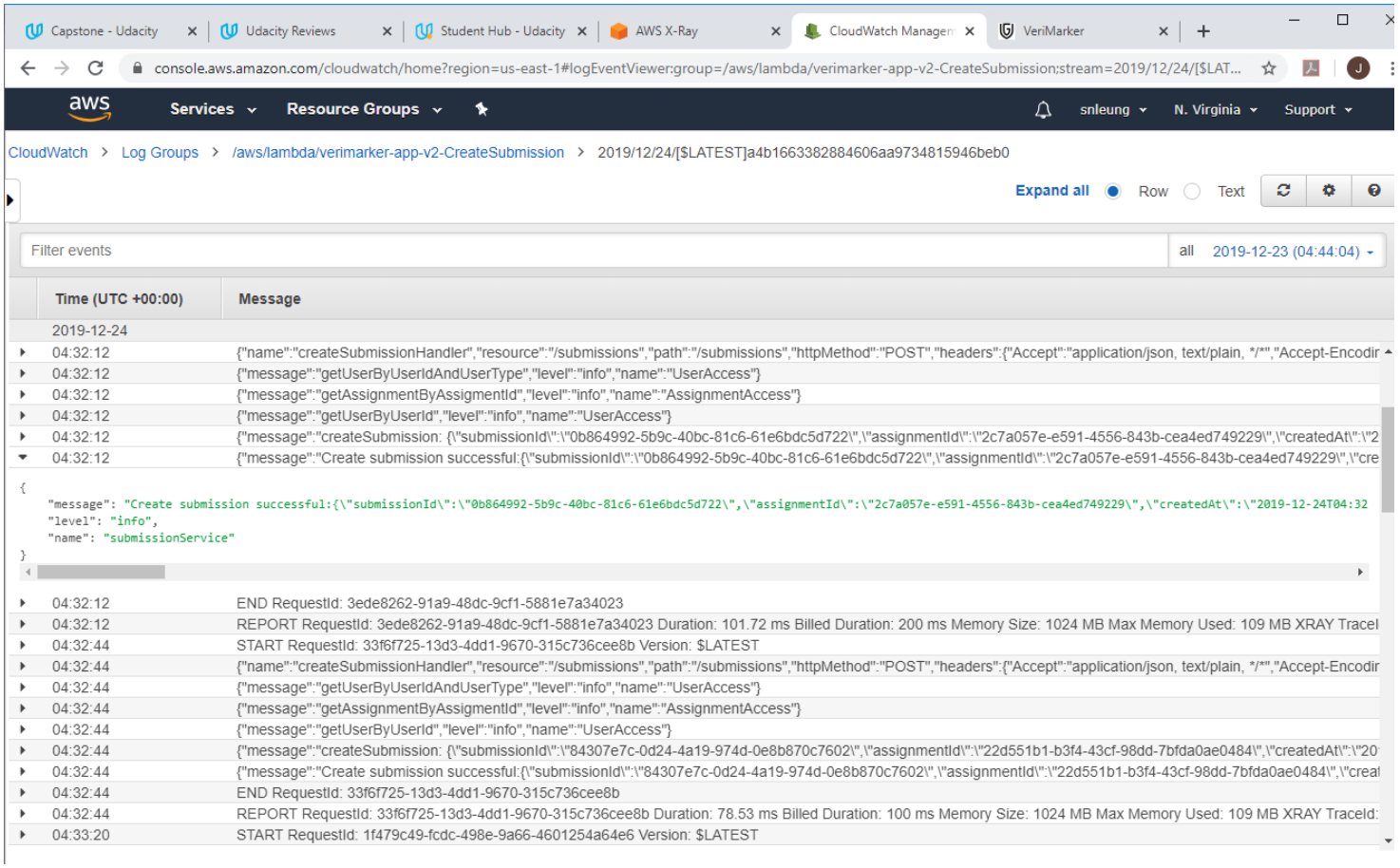
(continued to next page …)

- It has a sufficient amount of log statements

Our application uses the Winston logger which outputs the logs to AWS CloudWatch:

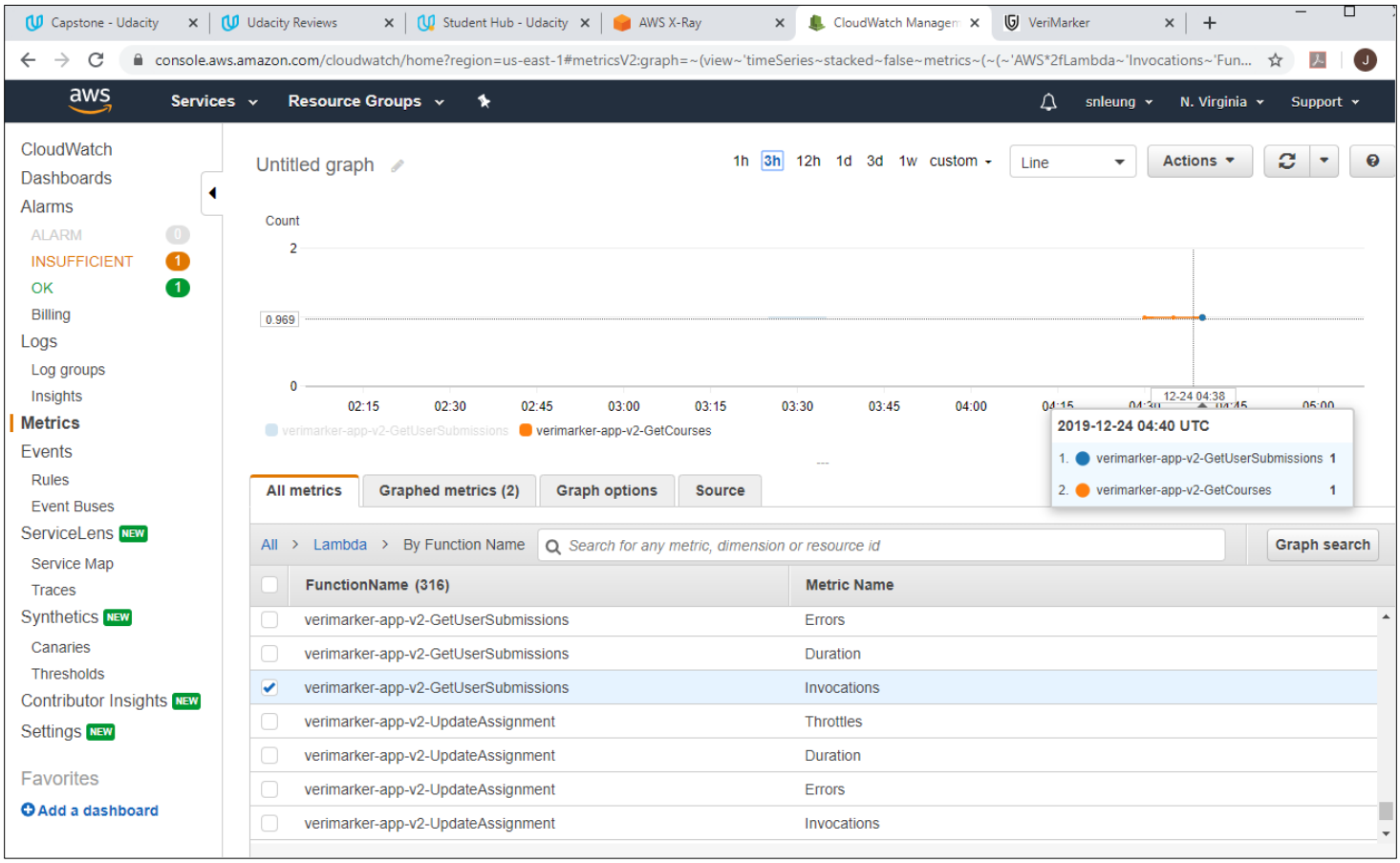


For example, for log group "/aws/lambda/verimarker-app-v2-CreateSubmission2", the following indicate the submission was created successfully (logged by submissionService). Also, note the assignment Id of this submission record (the student must select the assignment first before able to upload the submission to VeriMarker).



- It generates application level metrics

Under AWS CloudWatch, the application metrics of VeriMarker can be found by going to Metrics → Lambda → By Function Name. The following example selects Metric Name “Invocations” for FunctionName “verimarker-app-v2-GetUserSubmissions” and “verimarker-app-v2-GetCourses”:



(continued to next page …)

4.3.4 HTTP requests are validated

To reduce the number of unwanted Lambda function calls, incoming HTTP requests are validated using request validation in API Gateway. In serverless.yml:

```
custom:
  dynamodb:
    stages:
      - v2
    start:
      port: 8000
      inMemory: true
      migrate: true
  webpack:
    webpackConfig: ./webpack.config.js
    includeModules: true
  documentation:
    api:
      info:
        version: v1.0.0
        title: VeriMarker API
        description: Serverless application for VeriMarker, a plagiarism detection applica
tion for managing student's submissions
    models:
      - name: CreateUserRequest
        contentType: application/json
        schema: ${file(models/user/create-user-request.json)}
      - name: UpdateUserRequest
        contentType: application/json
        schema: ${file(models/user/update-user-request.json)}
      - name: CreateCourseRequest
        contentType: application/json
        schema: ${file(models/course/create-course-request.json)}
      - name: UpdateCourseRequest
        contentType: application/json
        schema: ${file(models/course/update-course-request.json)}
      - name: CreateAssigmentRequest
        contentType: application/json
        schema: ${file(models/assignment/create-assignment-request.json)}
      - name: UpdateAssignmentRequest
        contentType: application/json
        schema: ${file(models/assignment/update-assignment-request.json)}
      - name: CreateSubmissionRequest
        contentType: application/json
        schema: ${file(models/submission/create-submission-request.json)}
      - name: UpdateSubmissionRequest
        contentType: application/json
        schema: ${file(models/submission/update-submission-request.json)}
```

All of the lambda functions with method post or patch will have the RequestBodyValidator and requestModels defined. For example, in CreateSubmission (which is used by the student to upload submission):

```
  CreateSubmission:
    handler: src/lambda/http/submission/createSubmission.handler
    events:
      - http:
          method: post
          path: submissions
          cors: true
          authorizer: Auth0
          reqValidatorName: RequestBodyValidator
          documentation:
            summary: Create a new submission for the student
            description: Create a new submission for the student
            requestModels:
              'application/json': CreateSubmissionRequest
```

And in UpdateSubmission (which is used by the student to update the Submission References and Instructor to assign a score and add instructor comments to the submission):

```
  UpdateSubmission:
    handler: src/lambda/http/submission/updateSubmission.handler
    events:
      - http:
          method: patch
          path: submissions/{queryId}
          cors: true
          authorizer: Auth0
          reqValidatorName: RequestBodyValidator
          documentation:
            summary: Update existing submission of the student
            description: Update existing submission of the student
            requestModels:
              'application/json': UpdateSubmissionRequest
```

For the above two examples, let's take a look at the validation models of create-submission-request.json and update-submission-request.json:

(continued to next page …)

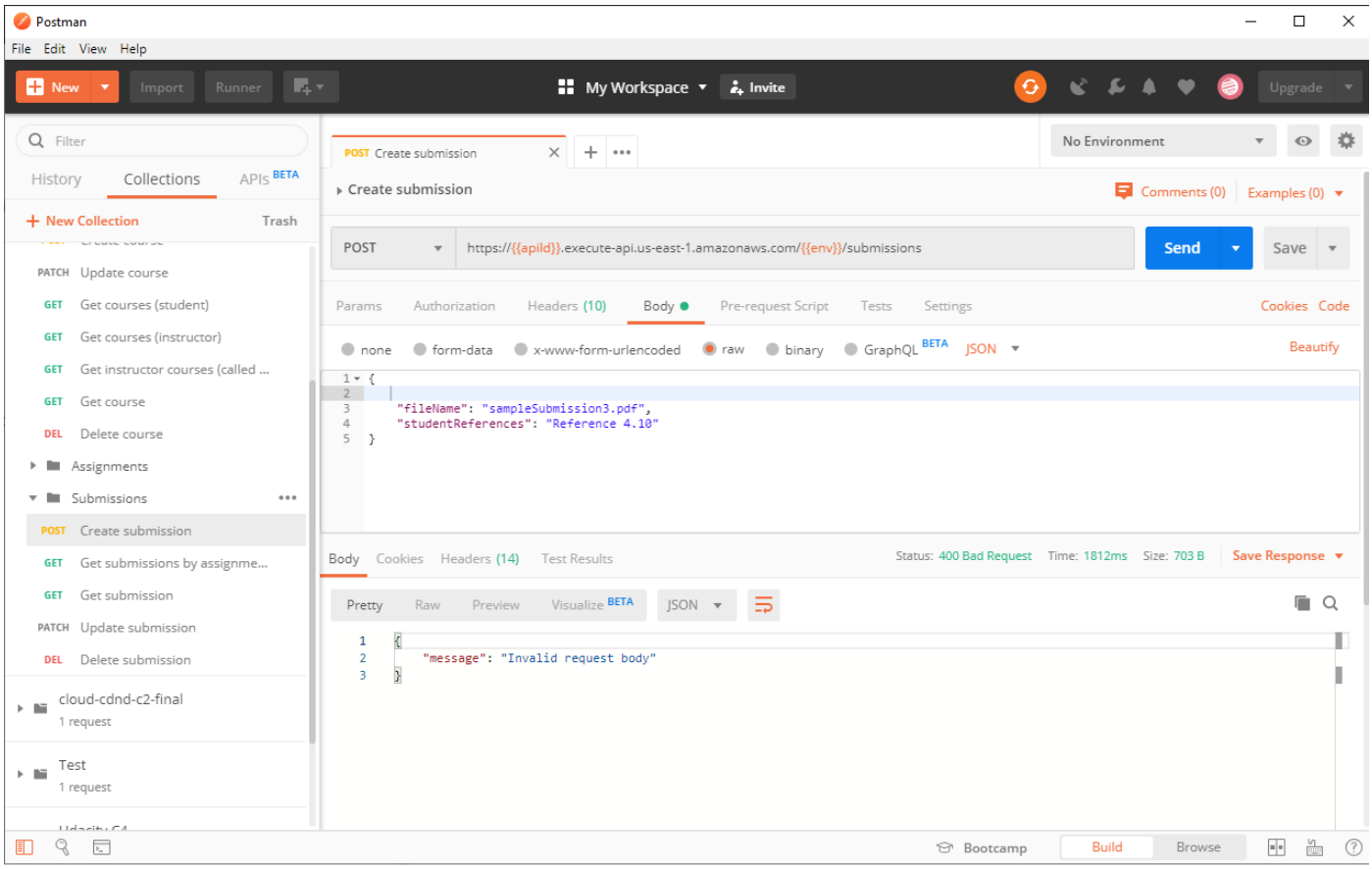
For the create-submission-request.json, all properties are required (mandatory):

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "createSubmissionRequest",
    "type": "object",
    "properties": {
      "assignmentId": {
        "type": "string"
      },
      "fileName": {
        "type": "string"
      },
      "studentReferences": {
        "type": "string"
      }
    },
    "required": [
      "assignmentId",
      "fileName",
      "studentReferences"
    ],
    "additionalProperties": false
}
```

For the update-submission-request.json, only the studentReferences is required. This is because instructorComments and studentScore are not defined initially when the submission was uploaded by the student (submission score and comments will be added later by the Instructor when the Instructor evaluate the student's submission).

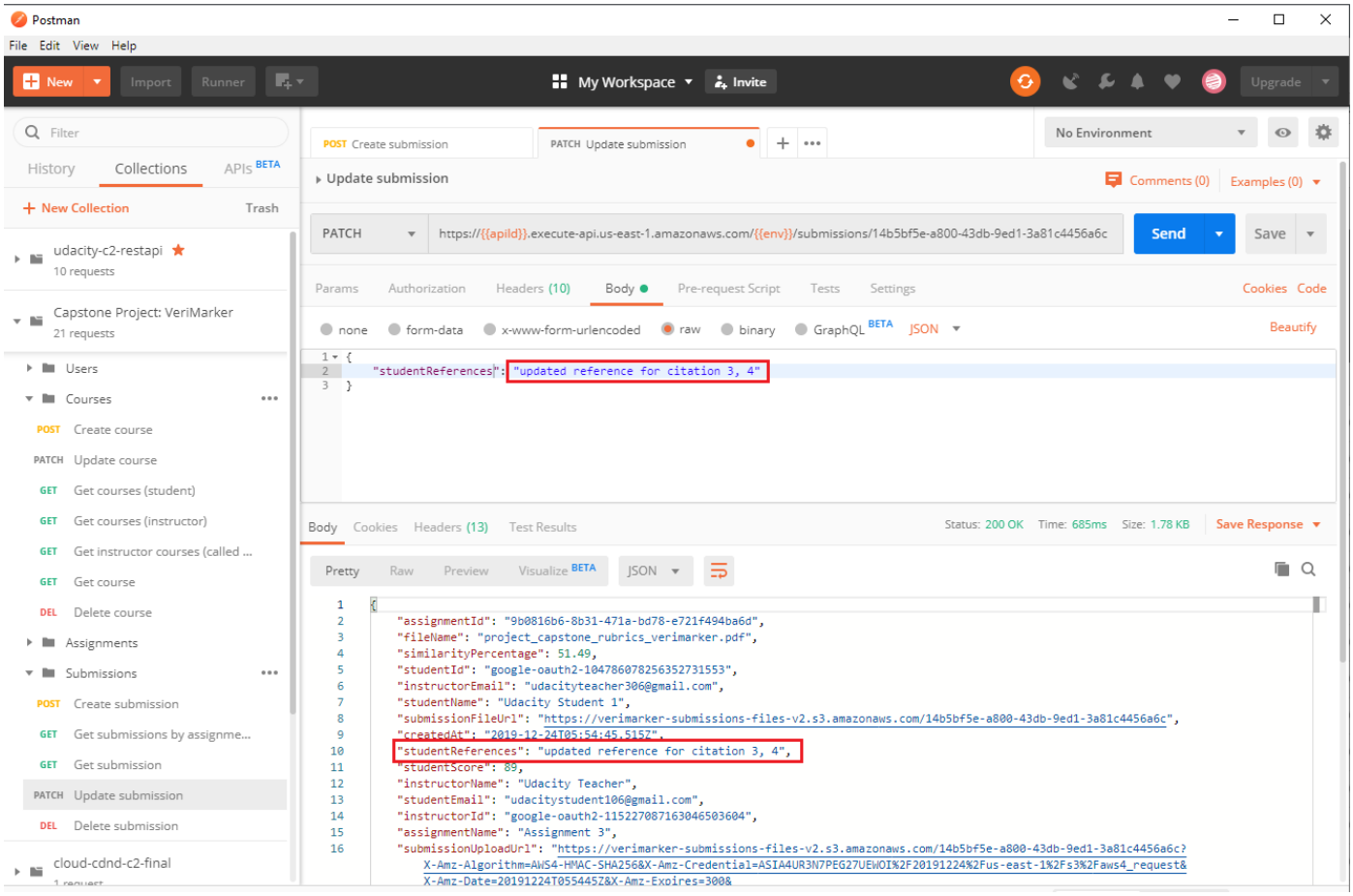
```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "updateSubmissionRequest",
    "type": "object",
    "properties": {
      "instructorComments": {
        "type": "string"
      },
      "studentScore": {
        "type": "number"
      },
      "studentReferences": {
        "type": "string"
      }
    },
    "required": [
      "studentReferences"
    ],
    "additionalProperties": false
}
```

In the first example, creating a submission request without the assignment Id will result in a response with code 400 Bad Request and message “Invalid request body”:

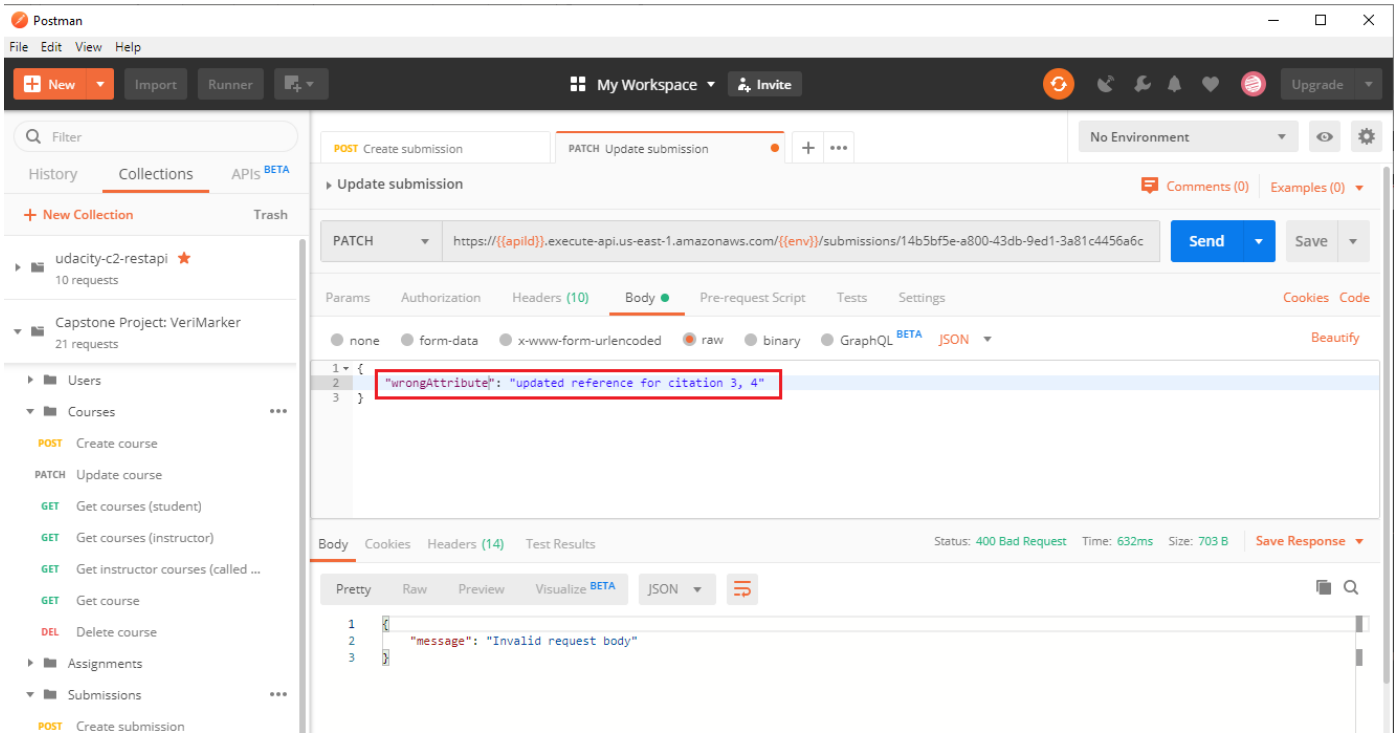


(continued to next page …)

In the second example, sending a PATCH request with only the studentReferences will successfully update the submission item without errors:



However, define a wrong attribute name in the patch upate submission request will result in a response with code 400 Bad Request and message “Invalid request body”:



4.4 Architecture

4.4.1 Data is stored in a table with a composite key.

1:M (1 to many) relationship between users and items is modeled using a DynamoDB table that has a composite key with both partition and sort keys. VeriMarker is heavily built upon the one to many (1:M) relationship exists in DynamoDB:

- **One** Assignment to **Many** Submissions
- **One** Student to **Many** submissions
- **One** Instructor to **Many** Courses
- **One** Course to **Many** Assignments

Please checkout the table definitions at the end of the serverless file. As an example, in the SubmissionsDynamoDbTable shown below, the submissions can be queried by the assignment Id and the student Id which is defined as the GlobalSecondaryIndexes:

```
    SubmissionsDynamoDBTable:
      Type: "AWS::DynamoDB::Table"
      Properties:
        AttributeDefinitions:
          - AttributeName: submissionId
            AttributeType: S
          - AttributeName: assignmentId
            AttributeType: S
          - AttributeName: studentId
            AttributeType: S
          - AttributeName: instructorId
            AttributeType: S
          - AttributeName: createdAt
            AttributeType: S
        KeySchema:
          - AttributeName: submissionId
            KeyType: HASH
          - AttributeName: createdAt
            KeyType: RANGE
        GlobalSecondaryIndexes:
          - IndexName: ${self:provider.environment.SUBMISSIONS_SUBMISSIONID_INDEX}
            KeySchema:
            - AttributeName: submissionId
              KeyType: HASH
            - AttributeName: createdAt
              KeyType: RANGE
            Projection:
              ProjectionType: ALL
          - IndexName: ${self:provider.environment.SUBMISSIONS_ASSIGNMENTID_INDEX}
            KeySchema:
            - AttributeName: assignmentId
              KeyType: HASH
            - AttributeName: createdAt
```

```
            KeyType: RANGE
          Projection:
            ProjectionType: ALL
        - IndexName: ${self:provider.environment.SUBMISSIONS_STUDENTID_INDEX}
          KeySchema:
          - AttributeName: studentId
            KeyType: HASH
          - AttributeName: createdAt
            KeyType: RANGE
          Projection:
            ProjectionType: ALL
        - IndexName: ${self:provider.environment.SUBMISSIONS_INSTRUCTORID_INDEX}
          KeySchema:
          - AttributeName: instructorId
            KeyType: HASH
          - AttributeName: createdAt
            KeyType: RANGE
          Projection:
            ProjectionType: ALL
      BillingMode: PAY_PER_REQUEST
      TableName: ${self:provider.environment.SUBMISSIONS_TABLE}
```

4.4.2 Scan operation is not used to read data from a database.

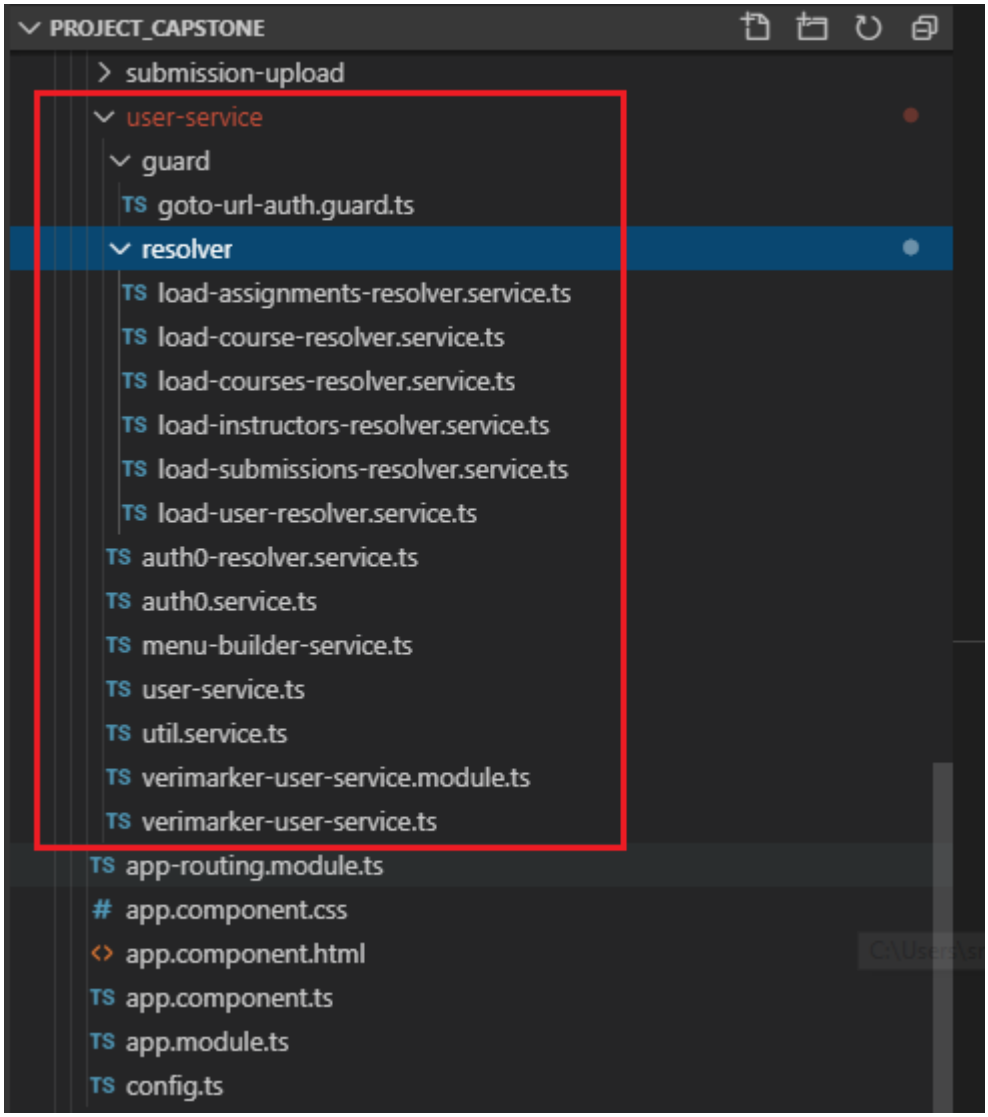
With the use of GlobalSecondaryIndexes , VeriMarker only uses the query method to retrieve records from DynamoDb. No scan method is ever used in the application.

(continued to next page …)

5. Appendix

5.1 Implementation of the Client

The Client is a webapp developed by using Angular and make heavily use of modules. In particular, the VerimarkerUserServiceModule is particularly important, and the following shows the services included in this module:



As seen from the above, the VeriMarker client app made use of the Angular Resolver (under the resolver folder), which issue HTTP requests to the AWS Lambda functions to get the data before the page is displayed to the user. This prevents the problem of "Flash of unstyled content" (FOUC) commonly found in most SPA applications.

The resolver (auth0-resolver.service.ts) is also used to listen on the callback called by Auth0, in order to determine whether the user should go directly to the Main Page of VeriMarker, or if the user is not registered in VeriMarker, go to the user registration page. Finally, the route guard (goto-url.auth.guard.ts) is used to determine whether the user is logged in (by checking on the user cookie which stored the user session JSON data of LoggedInUser.ts). If the user is already logged in, the user is permitted to go directly to the URL address, and if not, the user is redirected to the Login Page.

5.2 Linking Auth0 JWT token to VeriMarker User Account

It is found that although the idToken (JWT Token) of Auth0 can be different each time the user logged using Auth0, the decrypted JWT token, JwtPayload.sub, is a constant unique value for each Google user. As a result, this piece of information can be used to allow the Auth0 JWT token to link with a VeriMarker user account.

As mentioned before, the auth0-resolver.service.ts (in verimarker-school-login-routing.module.ts) listens on the callback Url after user logged in from Auth0:

```
  {
    // listen path: for dev = http://localhost:4200/school/auth0,
    // listen path: for prod = 'https://www.verimarker.com/school/auth0'
    path: verimarkerInjectors.get(URL_PATH_CONFIG).userAuth0CallBackPath.relativePath,
    resolve: {
      auth0ResolverService: Auth0ResolverService
    },
  },
```

In Auth0ResolverService, it delegates to the auth0.service.ts HandleAuthentication(), which obtains the idToken from auth0.parshHash. Using the idToken, the client queries the AWS Lambda function GetUser in order to get the VeriMarker registered user:

```
  handleAuthentication() {
    return this.auth0.parseHash( async (err, authResult) => {
      if ( authResult && authResult.accessToken && authResult.idToken ) {
        console.log('Access token: ', authResult.accessToken );
        console.log('id token: ', authResult.idToken );
        try {
          this.spinner.show();
          const headers = new HttpHeaders({
            'Content-Type':  'application/json',
            Authorization: `Bearer ${authResult.idToken}`
          });

          const user = await this.http.get<User>(`${apiEndpoint}/user`, { headers }).toPro
mise();
          this.spinner.hide();

          console.log(user);
          // user is registered in the system, fetch the user details
          // and store the user in the cookie as our session cookie
          if ( user.userId != null ) {
            this.userService.setLoggedInUser({
              authenticationState: AuthenticationStateEnum.Authenticated,
              userName: user.userName,
              email: user.email,
              userType: user.userType,
              accessToken: authResult.accessToken,
              idToken: authResult.idToken,
              userId: user.userId
            });
            // navigate to the user main page
```

```
            this.router.navigate( [ verimarkerInjectors.get(URL_PATH_CONFIG).userMainPage.
fullPath ] );
             ......................
```

A non-null value in the userId of the returned User indicates the user is already registered in VeriMarker. Thus, the Angular client will call UserService.setLoggedInUser, which stores the attributes of the returned user and the idToken/accessToken of Auth0 as an JSON object as a cookie:

```
    setLoggedInUser(loggedInUser: LoggedInUser ) {

        // cookie duration same as the Auth0 JWT duration
        this.cookieService.set( `${this.cookieName}-${authConfig.mode}`,
                                JSON.stringify(loggedInUser),
                                new Date( new Date().getTime() + 36000 * 1000 ),
                                '/' );

        const alreadyLoggedInUserStr = this.cookieService.get( `${this.cookieName}-
${authConfig.mode}` );
        console.log('alreadyLoggedInUserStr=' + alreadyLoggedInUserStr);

        this.loggedInUserObservable.next( loggedInUser );
    }
```

The LoggedInUser.idToken is used by VeriMarkerHttpClient, which is a wrapper of Angular HttpClient. Any HTTP requests with the AWS Lambda functions after user logged in will be done by the VeriMarkerHttpClient, whose primary purpose is to automatically set the idToken in the authorization header for each HTTP request.

Finally, the already registered user is navigated to the main menu.

If the user cannot be found (userId of the returned user from the AWS Get User method is null), the idToken and accessToken are passed to UserService.setRegisterNewUser. The application will redirect the user to the Registration Page which will use the idToken to POST the new user to the AWS Lambda function CreateUser:

```
.............................
      else {
            // navigate to register new user
            this.userService.setRegisterNewUser({
              authenticationState: AuthenticationStateEnum.NeedToCreate,
              idToken: authResult.idToken,
              accessToken: authResult.accessToken,
              userId: null,
              userType: '',
              email: '',
              userName: ''
            });

            let url = verimarkerInjectors.get(URL_PATH_CONFIG).userRegistrationPage.fullPa
th;
            url = url.replace(':userId', '0');
            this.router.navigate( [ url ] );
          }
```

user-registration.component (located in src/app/common-ui/component/user-registration):

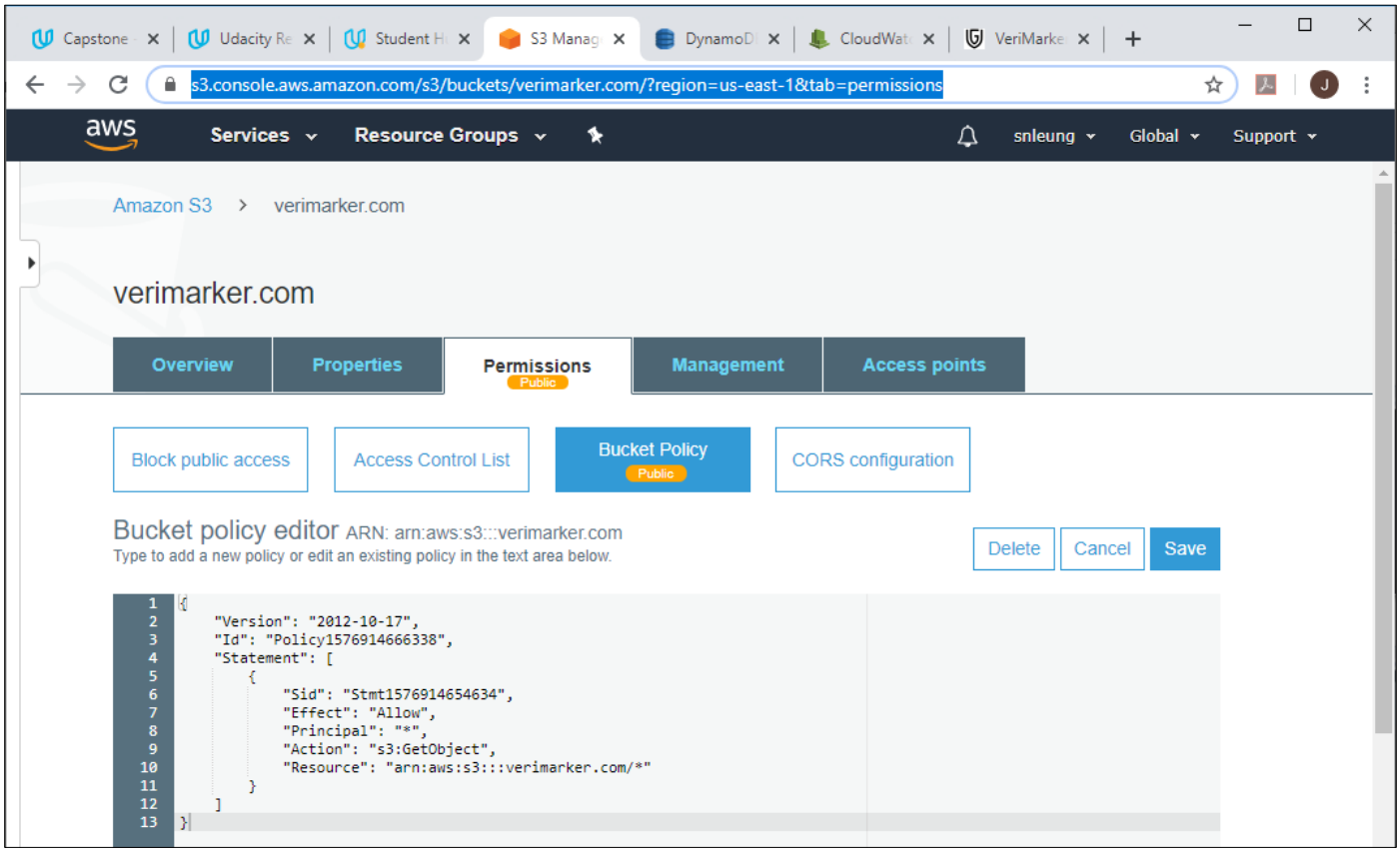
```
 async createNewUser() {
    const createUserRequest: CreateUserRequest = {
      userType: this.registerUser.userType,
      email: this.registerUser.email,
      userName: this.registerUser.userName
    };

    const headers = new HttpHeaders({
      'Content-Type':  'application/json',
      Authorization: `Bearer ${ this.registerUser.idToken}`
    });

    this.spinner.show();
    try {
      const user = await this.http.post( `${apiEndpoint}/users`, createUserRequest, { head
ers } ).toPromise() as User;
      this.spinner.hide();

      // for create new user when login to auth0, set the newly created user as our logged
 in user and create the session cookie
      if ( this.registerUser.authenticationState === AuthenticationStateEnum.NeedToCreate
) {
        this.alertDialogService.openDialog({
          title: 'Register New User',
          message: 'Successfully registered as a new user.',
          dialogType: 'OKDialog'
        }).then( res => {
          this.userService.setLoggedInUser({
            authenticationState: AuthenticationStateEnum.Authenticated,
            userName: user.userName,
            email: user.email,
            userType: user.userType,
            accessToken: this.registerUser.accessToken,
            idToken: this.registerUser.idToken,
            userId: user.userId
          });
        });
      }
    } catch (e) {
      this.spinner.hide();
      console.log(e);
    }
  }
```

5.2 Deployed Angular Client to AWS S3 and CloudFront using route 53

The Angular client was deployed to https://www.verimarker.com

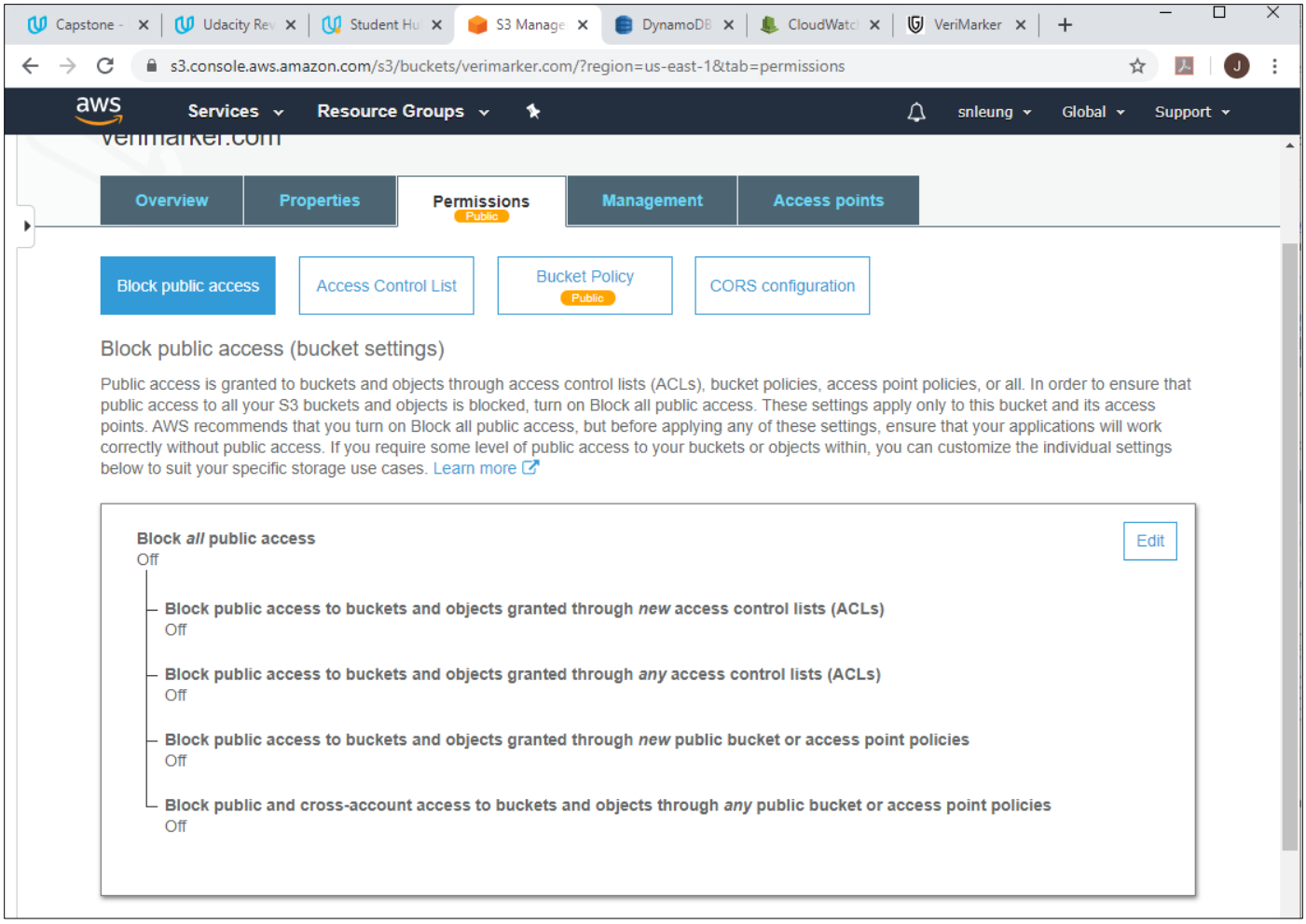
Here are few important notes when deployed the Angular client to AWS S3 / CloudFront:

1. Create S3 bucket www.verimarker.com
2. In the S3 bucket, ensure to include the following bucket policy:

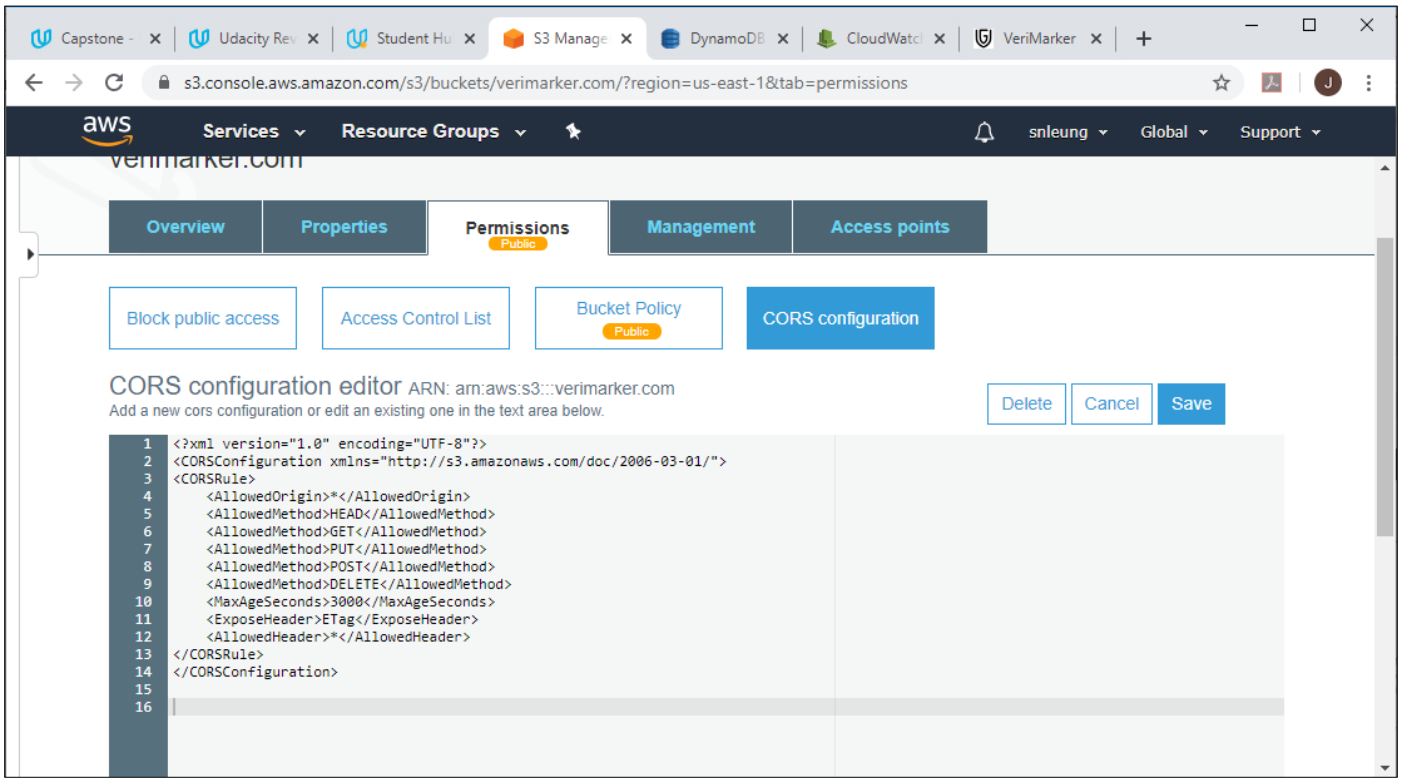


3. In the S3 bucket, ensure to remove all check marks under Block public access:

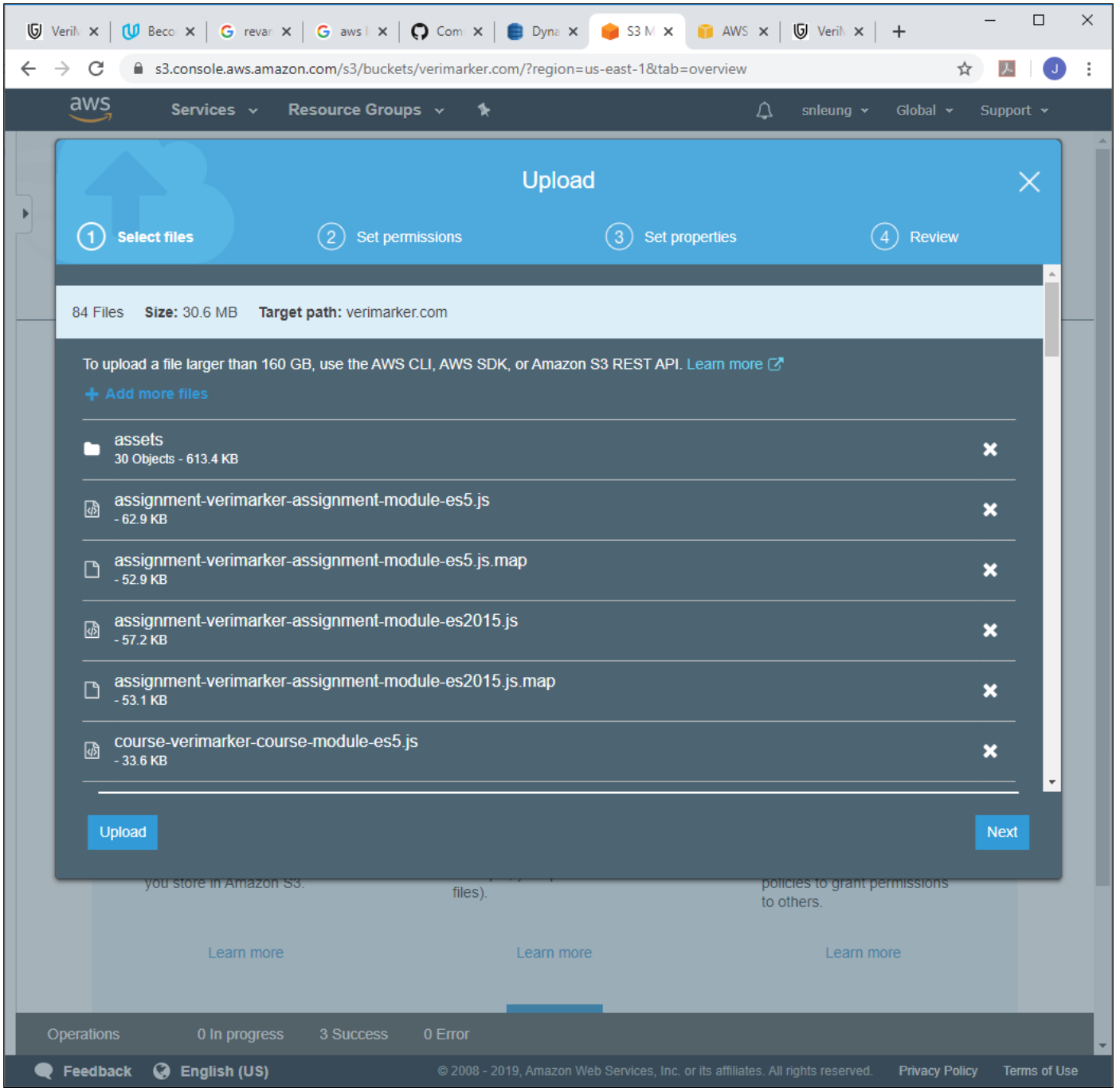
(continued to next page …)



4. Enable CORS configuration in S3:

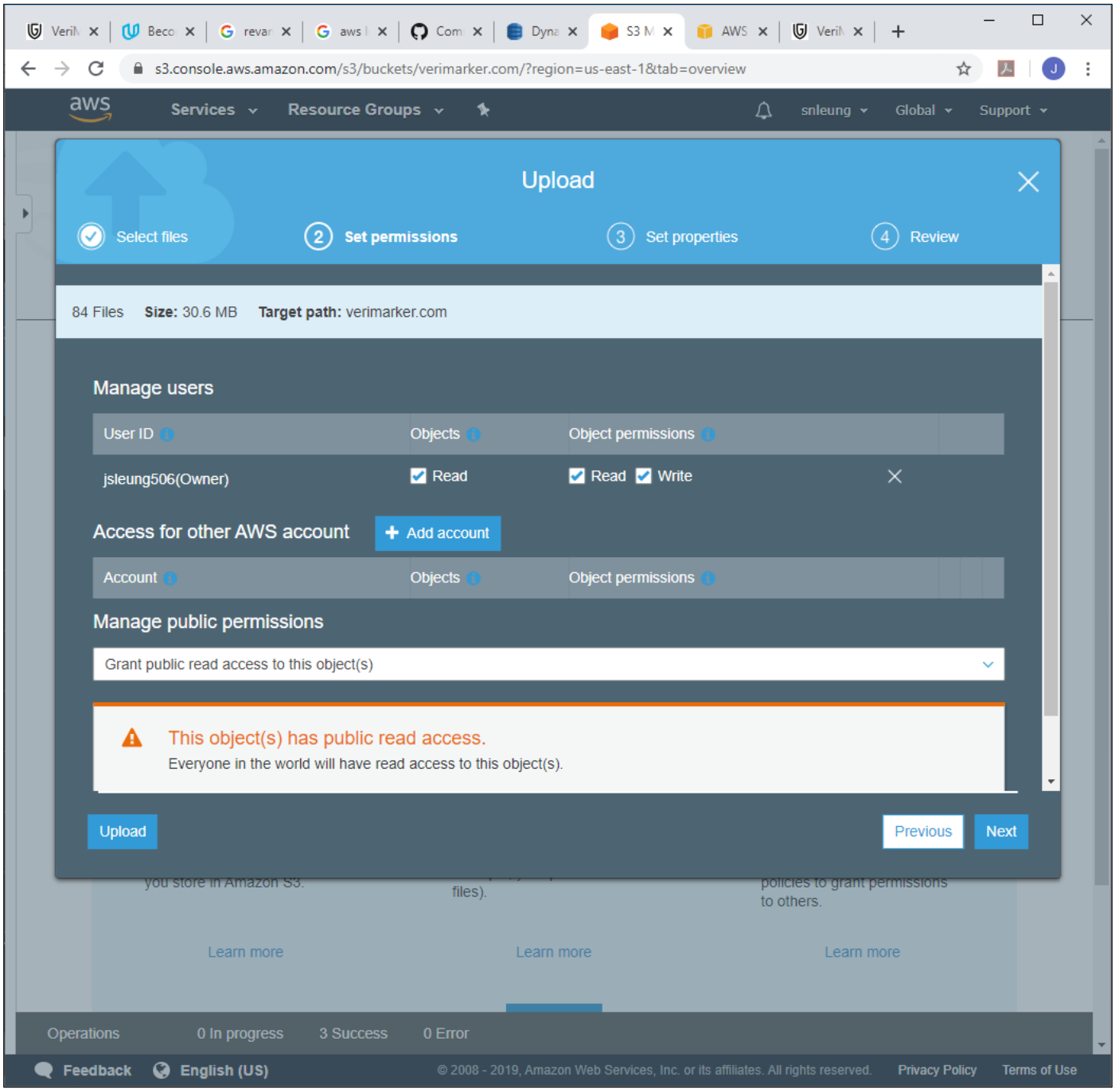


5. Go to Visual Studio Code. In client folder, execute ng build.
6. This will build the Angular application located in client/dist/verimarker.
7. In S3, select the bucket www.verimarker.com. In Overview, click on Upload.
8. Drag the files inside the dist folder to the S3 Upload dialog:



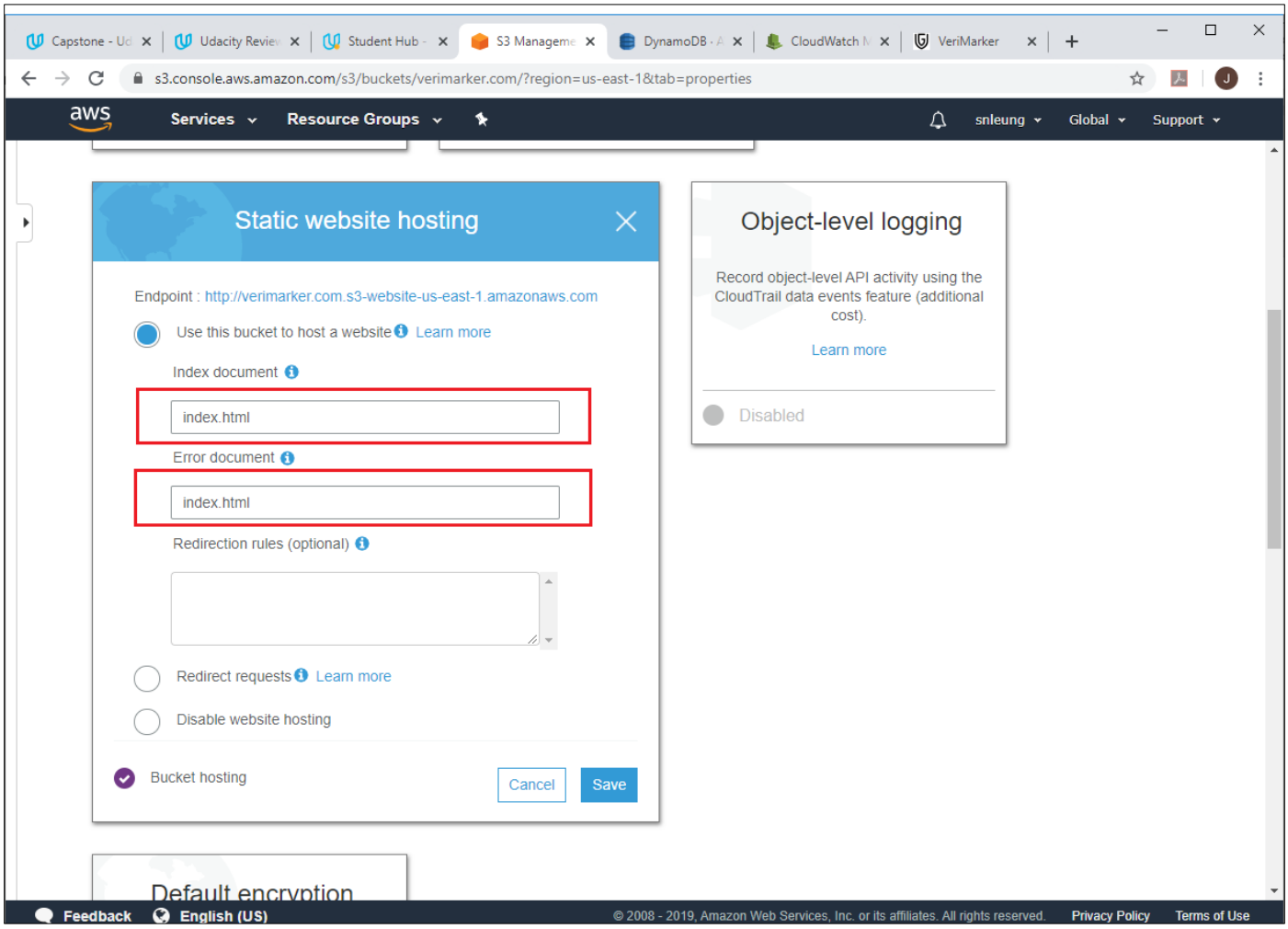
9. Click Next.
(continued to next page …)

10. Select “Grant public access to this object(s)” and Click Next to complete the upload.



(continued to next page …)

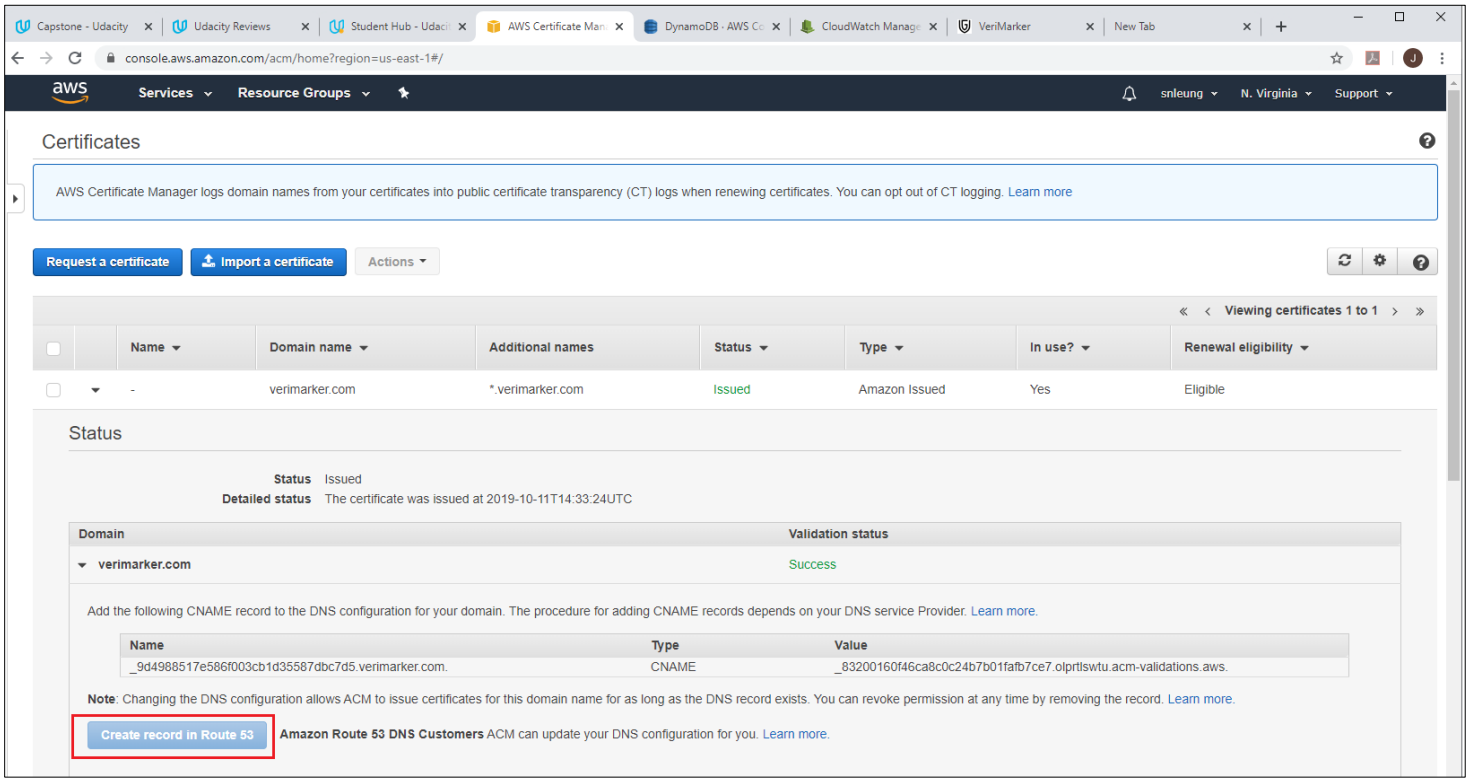
11. Go to Properties. Select Static Web Site Hosting.
12. Select Use this bucket to host a website. Enter index.html for BOTH the Index document and the Error document:



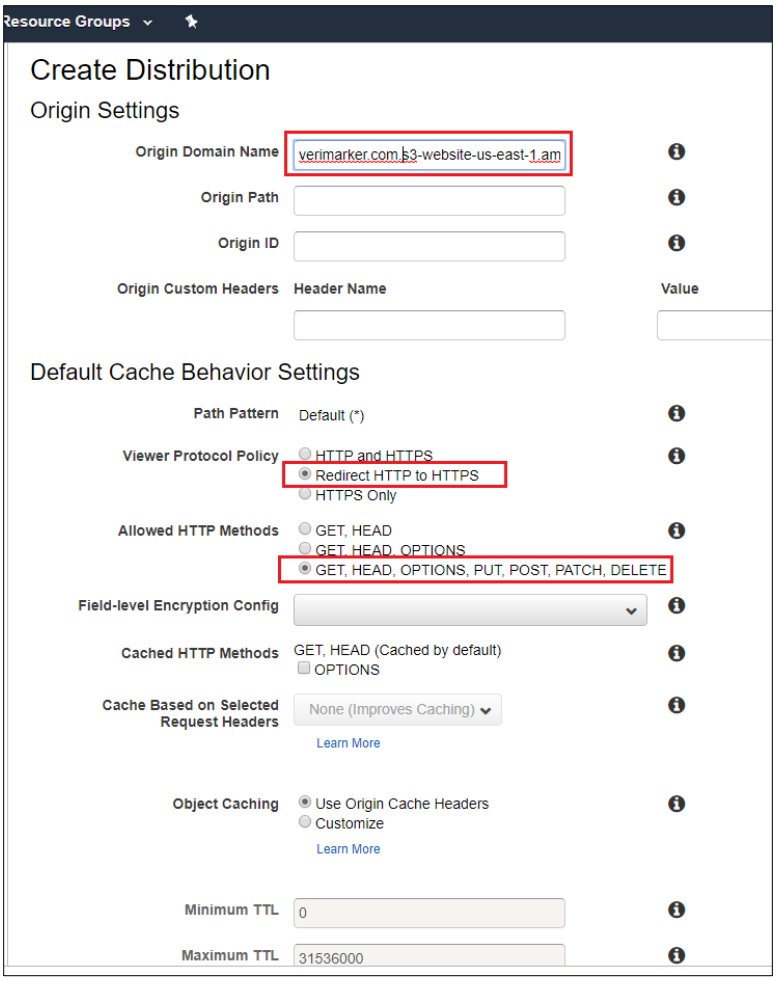
13. Click Save. Go to Route 53. In registered domain, purchase a domain (verimarker.com).

(continued to next page …)

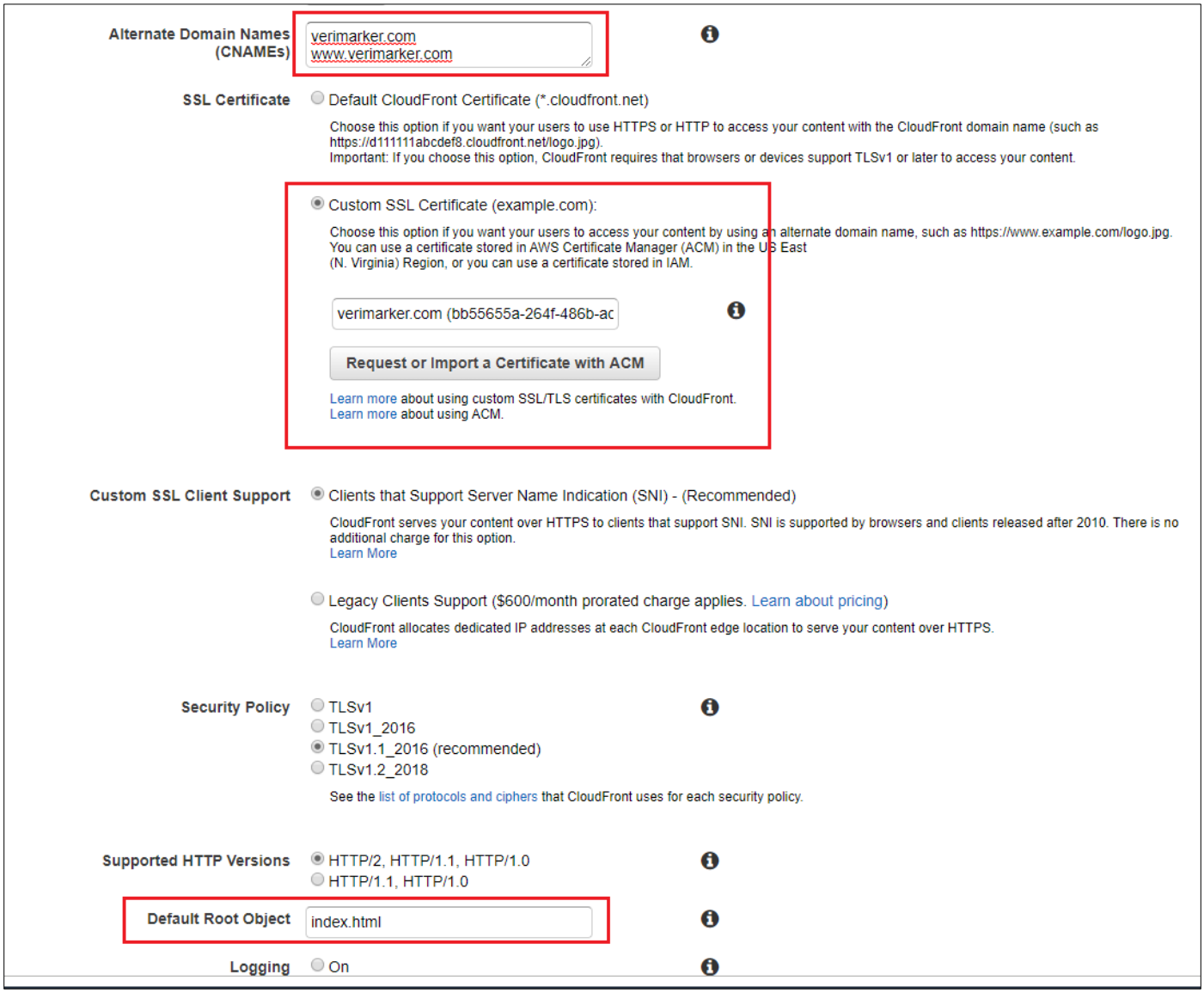
14. In Certificate Manager, click on Create record in Route 53 for both verimarker.com and *.verimarker.com



15. Go to CloudFront, Create Distribution (for Web):



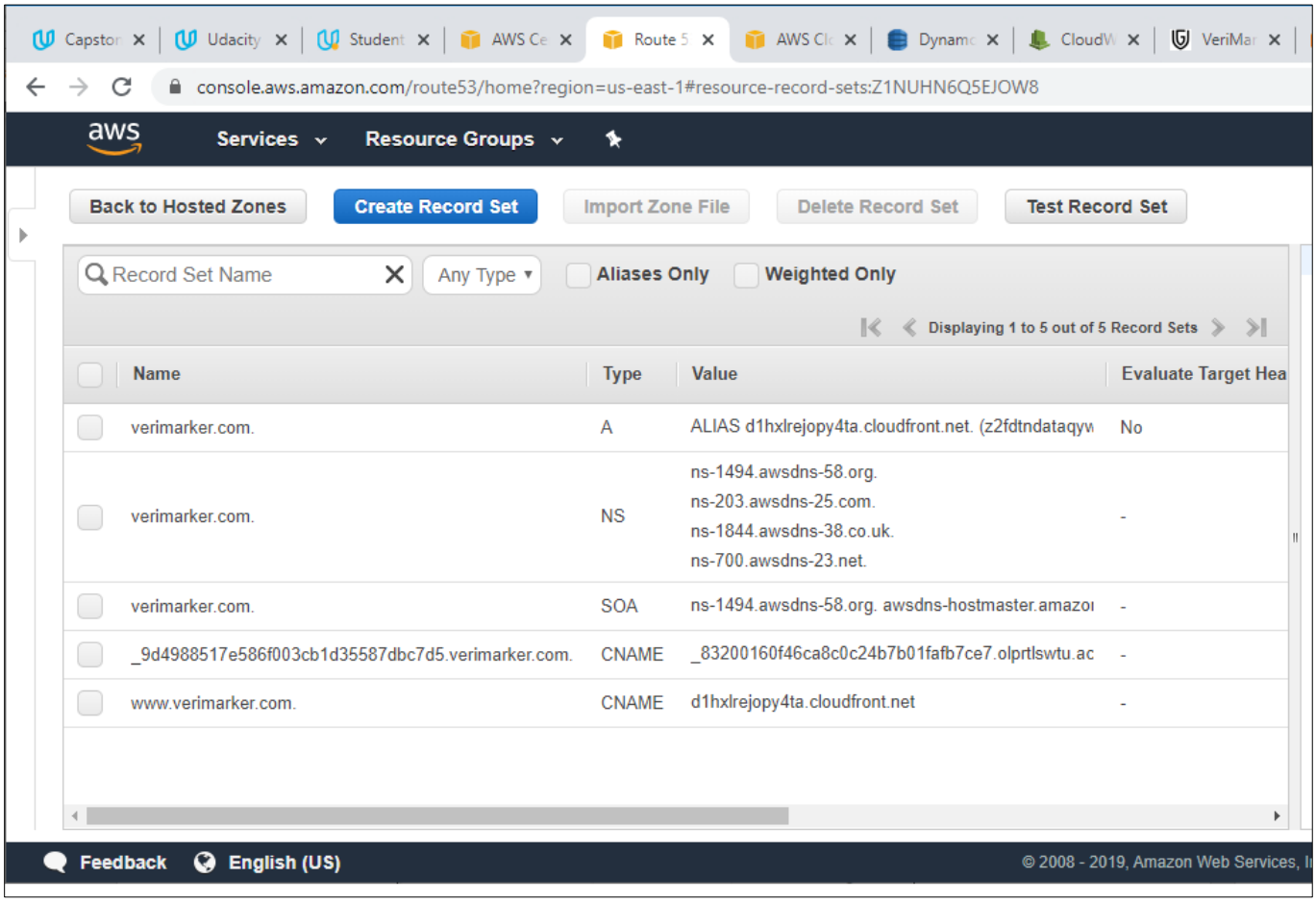
16. For Origin Domain Name, MANUALLY typped in verimarker.com.s3-website-us-east-1.amazonaws.com .
17. Select Redirect HTTP to HTTPS.
18. Allowed HTTP Methods: GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE.



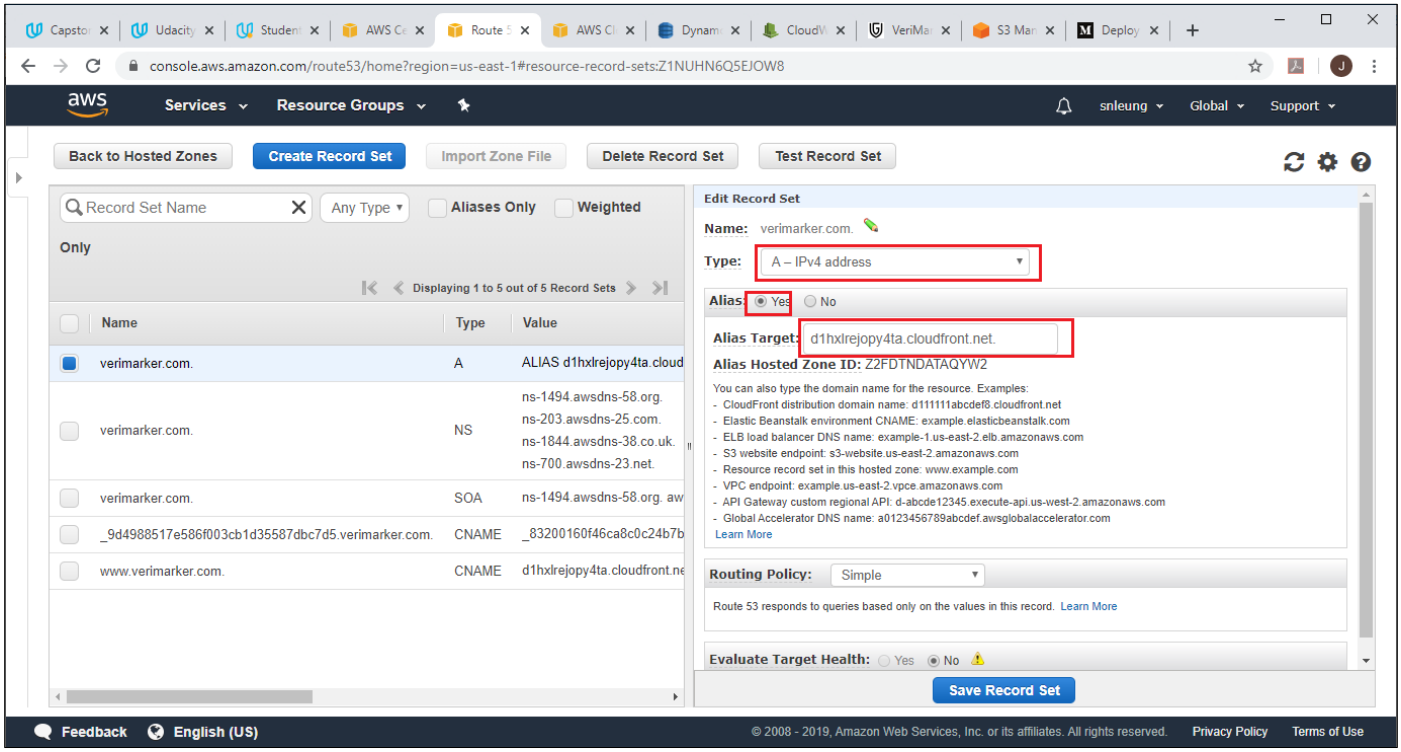
19. For Alternate Domain Names, enter verimarker.com and www.verimarker.com
20. SSL Certificate: select Custom SSL, and select verimarker.com in the dropdown.
21. Default Root Object: index.html.
22. Remove checkbox for Enable IPv6 (at the bottom of the page).
23. Click Create Distribution.

(continued to next page …)

24. Go to Route 53 (The following already show the A and CNAME records for CloudFront distribution were already created):



25. Create 'A' Record Set: Select Alias = 'Yes', Alias Target: d1hxlrejopy4ta.cloudfront.net. (the CloudFront distribution of VeriMarker):



26. Create CNAME record set. Enter value: d1hxlrejopy4ta.cloudfront.net.