任务分工

Du Liang

**Course roster download** – this action, implemented by the GET /courses/{id}/roster endpoint, allows certain authorized users to download a CSV-formatted roster for a specific course.  The roster will contain a list of the students currently enrolled in the course, in CSV format, e.g.:  
  
"abc123","Jane Doe","doej@oregonstate.edu"  
"def456","Luke Skywalker","skywallu@oregonstate.edu"  
...  
  
Importantly, this file must be generated by the API, based on the list of enrolled students stored in the database.  There are several alternative ways this can be accomplished. For example, each time the roster is requested, it can be generated on the fly based on appropriate database queries before being sent back to the client.  Alternatively, the roster could be pre-generated automatically whenever an update is made to a course’s enrollment, and requests for the roster can simply send back this pre-generated file.

**Course information fetching** – this action, implemented by the GET /courses and GET /courses/{id} endpoints, allows users to see information about all Courses or about a specific Course.  Note that the information included by both of these endpoints should not return information about a Course’s enrolled students or its Assignments.  Instead, that information can be fetched by the GET /courses/{id}/students and GET /courses/{id}/assignments endpoints, respectively.

**Other Under The /courses/\***

Feng Liang

**Assignment submission creation** – this action, implemented by the POST /assignments/{id}/submissions endpoint, allows authorized student Users to upload a file submission for a specific assignment.  Importantly, the file uploaded for each Submission must be stored by the API in such a way that it can be later downloaded via URL.  Specifically, when storing the submission file, the API should generate the URL with which that file can later be accessed. This URL will be returned along with the rest of the information about the Submission from the GET /assignments/{id}/submissions endpoint.

**Other Under The /assignments/\***

The JWT verify Model.

IP address-based rate limiting

Zhuohong

**User data fetching** – this action, implemented by the GET /users/{id} endpoint, allows Users to see their own data.  Importantly, only a logged-in User can see their own data.  The data returned by this endpoint should also include the list of classes the User is enrolled in (for student Users) or teaching (for instructor Users).

The User Authaction for the JWT

**Other Under The /users/\***

**The User Login PlatForm**

**The User Create Link**

All files work under the express likely the assignment4

Course router using the course.js

Assignment using the Assignments.js

Users using the users.js

User Grade For Jwt(auth js will return it in req.usertype):

admin

instructor

student

The model for the auth is called auth.js

const jwt = require('jsonwebtoken');

const secretKey = 'RandomString';

exports.generateAuthToken = function (userId,type) {

const payload = {

sub: userId,

type: type

};

const token = jwt.sign(payload, secretKey, { expiresIn: '24h' });

return token;

};

exports.requireAuthentication = function (req, res, next) {

const authHeader = req.get('Authorization') || '';

const authHeaderParts = authHeader.split(' ');

const token = authHeaderParts[0] === 'Bearer' ? authHeaderParts[1] : null;

try {

const payload = jwt.verify(token, secretKey);

req.user = payload.sub;

req.usertype=payload.type;

next();

} catch (err) {

console.error(" -- error:", err);

res.status(401).send({

error: "Invalid authentication token provided."

});

}

};

The Files Storage using the GridFS and the User info storage using the Mysql. The file and the info related by the \_id in GridFs.