GradeScope

Role

A guest can:

1. Sign in and sign up
2. Sign up an instructor account via email
3. Sign up a student account via course entry code
4. Sign in an account via registered email and password
5. (Extension) Link to a Google account for fast log in

A student can:

1. Join a course via entry code
2. View all the courses they have joint
3. Apply a rescore request for a single query in a submission
4. Resubmit a submission (overwrite previous one)
5. Check their submission history within an assignment
6. Download graded copy
7. Receive Email
8. Change profile
9. Log out

An instructor can:

1. Create a new course
2. Change the details of a course
3. Create a new assignment
4. Change the assignment information
5. Score a submission
6. Check submission history in an assignment for each student
7. Publish scores to student
8. Unpublished scores
9. Change the role in a course where they play an instructor role
10. Reply a rescore request
11. Change profile
12. Log out

API Doc

Account CRUD

~GET /account/login body = {accountEmail, accountPwd} //login, return a JWT including user entity, excluding accountPwd, secret~

~POST /account body = {accountEmail,accountPwd,accountName,role} // register, send an notification email, insert an document in email || Email delayed, Solution: use Amazon lambda and Amazon SMS to build a simple mail server.~

~PUT /account/:id body = {updatedInfo} //change account settings~

~GET /verify/:id body = {} //verify the availability of link and change the status of account to verified~

~PUT /enroll body={accountId, courseEntryCode} //update reference field for both collections.~

~GET /course/:id/accountList/page/:pageNum body={} //get some rosters for a course (teacher-course-rosters)~

GET /account/:id/courseList body= {} // get a list of course from a specific user

Course CRUD

~GET /courseList body={} //get All course~

~GET /course/:id body = {} // get information of a course~

~POST /course body= {courseName, courseDesc, startTerm, startYear, createdBy} //Create a course.~

~PUT /course/:id body={updatedInfo} // change the information for a course (change course setting, add material links, add or modify announcement)~

~GET /account/:id/courseList body={} // get all the courses for a user by semester~

Assignment CRUD

~GET /assignment/:id body={} // get info for an assignment~

~GET /course/:id/assignmentList body={} //get all the assignments for a course (student course dashboard)~

~POST /course/:id/assignment body={assignmentName,assignmentDesc,releaseDate,dueDate,delayDate} //create an assignment for a course~

~PUT /assignment/:id body={updatedInfo} // change the information for an assignment~

Submission CRUD

~GET /course/:id/assignmentList/submissionList body={} //get all the assignments for a course, including all submission information for these assignments (teacher course dashboard)~

~POST /assignment/:id/submission body={submitTime, submittedBy, uploadedFile} //query all the submissions under this assignment and student, find the latest submissions id as prevSubmission, forward the file to file server and get the file URL as answerURL field, insert a document into submission and update the latest submission array for this assignment document~

~PUT /submission/:id body={updatedInfo} //change the information for a submission (mark a submission, assign a submission to a teacher)~

~PUT /assignment/:id/submission/allocate body=[{accountId (non-student), allocatedNum}] //query all the unassigned submissions for an assignment, automatically allocate to teacher.~

GET /assignment/:id/submissionList body={} // get all the submissions for a specific assignment (useless)

GET /assignment/:assignmentId/account/:accountId/submission/history body={} //get submission for a user in an assignment

Account

```
JSON: {
  _id: accountId,
  accountEmail: "", //index, unique
  accountName: "FirstName|LastName",
  accountPwd:"",
  secret:"",
  role: "" in ("student", "teacher"),
  courses:[course.courseEntryCode],
  //extension function
  notificationType:1-3
  status: "" in ("pending", "verified", "deleted")
}
```

Email Notification

```
JSON: {
  _id: hashedLinkId, //easygrade.com/verify/:id
  accountId: account.accountEmail,
  expire: Date() (now + 15 mins),
}
```

Course

```
JSON: {
  _id: courseId,
  courseEntryCode: "", //automatically generated, index, unique
  createdBy: account.accountEmail (non-student),
  courseName: "",
  courseDesc: "",
  startTerm:"" in ("Mar", "Jun", "Sep", "Dec"),
  startYear:"",
  rosters: [account.accountEmail],
  isDeleted: boolean,
  announcements:[{
    title: '',
    content: '',
  }],
  material:[{  //not required
    title: "",
    link: "",
  }],
}
```

Assignment

```
JSON: {
  _id: assignmentId,
  courseId: course._id,
  assignmentName:"",
  assignmentDesc: "",
  releaseDate: Date(),
  dueDate: Date() (>releaseDate),
  delayDue: Date() (>=dueDate),
  // if cannot find submission for student, status is unsubmitted.
  status: 1-4 (ongoing, dued, allocated, published),
  totalMark:0-100,
  attachments:[{
    filename:"",
    fileURL:"",
  }],
  submission:[submission._id (latest)]
}
```

Assignment Criteria Template (Subset One2one) Priority Low!

```
JSON: {
  _id: febTemId,
  assignmentId: assignment._id,
  criteria:[{
    criteriaDesc:"",
    maxMark: int (1-100), // markrange [0, mark]
  }],
}
```

Submission Feedback Priority Low!

```
JSON: {
  _id: feedbackId,
  criteria febTem._id,
  submissionId: submission._id,
  criteria:[{
    mark: double (0-1), // markrange [0, maxMark*mark]
  }],
  feedback: "",
}
```

Submission

```
JSON: {
  _id: submissionId,
  prevSubmission: submissionId,
  submitTime: Time() (< assignment.dueDate),
  submittedBy: [{accountId (student)}],
  assignedTo: accountId (non-student),
  markedBy: accountId (non-student),
  gradedMark: 0-100,
  // when there is rescore request, reset it as true
  isUngraded: boolean,
  isPublished: boolean,
  answerURL: "",
```

RescoreRequest Priority Low!

```
JSON:{
  _id: rescoreStatusId,
  requestContent:"",
  requestDate: Date(),
  replyContent:"",
  replyDate: Date() (> requestDate),
  requestStatus: status in {"processed", "pending"},
}
```

Extension Request Priority Low!

```
JSON:{
  _id: extensionRequestId,
  assignmentId: assignment._id,
  requestBy: account.accountEmail,
  requestContent:"",
  requestDate: Date(),
  replyContent:"",
  replyDate: Date() (> requestDate),
  requestStatus: status in {"processed", "pending"},
  extendedDate: Date() (>= assignment.dueDate), //required: false
}
```

Challenges

- due date notification
- Calendar selector
- PDF editor

One to many (embedded document)

```
db.one.insertOne({
  _id: ObjectId(1),
  oneInfo: 'xxx',
})
db.many.insertOne({
  _id: Object(1),
  oneId: one._id, // FK
  manyInfo: 'xxx'
})

db.one.aggregate([
  {
    $addFields: {
      id_str: {$toString: "$_id"}
    }
  },
  {
    $lookup: {
      from: 'many',
      localField: 'id_str',
      foreignField: 'oneId',
      as: 'one2Many'
    }
}])
```

One to many (document reference)

```
db.one.insertOne({
  _id: Object(1),
  oneInfo: 'xxx',
  manyRef: [{
    manyId: many.id //FK
  }]
})

db.many.insertOne({
  _id: ObjectId(1),
  manyInfo: 'xxx'
})

db.one.aggregate([
  {
    $addFields: {
      "manyRef.objectId": {$toObjectId: "$manyRef.manyId"}
    }
  },
  {
    $match: {_id:1}
  }
  {
    $lookup: {
      from: 'many',
      localField: 'manyRef.objectId',
      foreignField: '_id',
      as: 'one2Many'
    }
  },
  {
    $project: {'one.manyRef': 0}
  }
])
```