



**CODEJUDGE**

**Fall 22**

Aravind Badavath  
Chetan Nagavathi Chendrayudu  
Gokul Sai Doppalapudi  
Jeya Krishna Chandu Akula  
Rajendra Thottempudi  
Shyam Prasad Nagulavancha

# Contents

## [Contents](#)

## [Overview of project implementation](#)

## [User stories](#)

[Completed user stories](#)

[Completed bugs](#)

[Completed story points](#)

[Iceboxed user stories](#)

## [Modifications to legacy project](#)

[Schema changes](#)

[Columns added](#)

[Tables created](#)

[Tables modified](#)

[Relationships added](#)

[Migration changes](#)

[Migrations added:](#)

[User profiling](#)

[UI for different users](#)

## [Roles of team members](#)

## [Iterations & Customer meetings](#)

[Iteration 0 + Meeting 0](#)

[Iteration 1 + Meeting 1](#)

[Iteration 2](#)

[Iteration 3 + Meeting 2](#)

[Iteration 4 + Meeting 3 + Meeting 4](#)

[Iteration 5 + Meeting 5 + Meeting 6](#)

## [BDD/TDD testing](#)

## [Git overview - Branches and Releases](#)

## [Local Deployment](#)

## [Heroku release process](#)

[Overview](#)

[Issues faced](#)

## [Software tools used](#)

[Overview](#)

[Issues](#)

[Benefits](#)

[Testing - cucumber and Rspec](#)

[Git repo contents](#)

[Project links](#)

## **Overview of project implementation**

CodeJudge is a legacy project that is a site for Texas A&M instructors to assign automatically judged programming problems to their students. While it provides some of the core functionalities, we are asked to implement some new features to enhance the usability of the CodeJudge application. Some of the requirements from the customer include user profiling, student grouping, assigning questions to a class, A student's grade, a solution to a problem, access code to join a class, tagging functionality in questions, Bulk upload of test cases and statistics for student groups

We have started with the most dependent story (user profiling), which is necessary for almost all of the new features to be implemented. For that, we have changed the necessary backend schema and created new UI screens for every user type. Following that, we concentrated on student grouping, classifying questions, and so on. We have accomplished almost all the functionalities, but due to time constraints, we couldn't complete bulk uploads of test cases and statistics on student groups. Regular meetings with clients helped us concentrate on the high priority stories and kept us on the right track.

## **User stories**

### ***Completed user stories***

1. Schema changes for the database - [183400457](#) (3 points) :  
One of our first tasks was to make changes to the existing schema to support all the upcoming features and functionalities of user profiles, user groups and the questions created by the instructor.
2. Add a new UI page for instructor view - [183400481](#) (3 points) :  
This story targets at designing and implementing a new UI for instructors which will be their default/home screen. It should support the feature of adding/removing/editing the questions that they have created.











3. Add a new UI page for admin view - [183400471](#) (3 points) :  
Design and implement a new UI for administrators which will be their default/home screen. It should support the feature of approving/rejecting a request made by the instructor when they sign up in the application.
4. Navigate the user to their default home page - [183644845](#) (3 points) :  
When a user logs into the Code judge application, We need to navigate the user to their default home page based on their profile.
5. Modify Backend service to support recent UI changes. - [183645012](#) (3 points) :  
Make necessary changes to the backend service by developing new functionalities to support the recent UI changes related to the default home page of instructor, administrator and student.
6. Rectify migration history and create new migrations - [183726857](#) (2 points) :  
Change existing migrations to support the current codebase and add new migrations to support changed schema.
7. Database schema and ER model for user profiles - [183726860](#) (2 points) :  
Make schema and appropriate association changes to the existing schema.
8. Add drop down to select different user types in registration page - [183726926](#) (1 point) :  
Design and implement a new Drop down UI in User Registration Page to support different user types (Instructor, Student). This drop down will help admin/instructor to approve/reject the requests from instructor/students registered with codeJudge.
9. Create a new UI for a class to support list of students - [183744257](#) (3 points):  
When clicked on a particular class, A new view for the class should be opened which should contain the list of students for a class and a list of problems for that class. There should be CRUD options available for students.
10. Instructor should be able to create or modify problems - [183784637](#) (3 points):  
Instructors should be able to perform CRUD operations in problems tab of instructor login page
11. Instructor should be able to create classes or groups - [183784630](#) (3 points)  
Instructor should be able to perform CRUD operations on class
12. Create a new view and UI for a class to support list of questions - [183744209](#) (3 points) :  
When clicked on a particular class, A new view for the class should be opened which should contain the list of students for a class and a list of problems for that class. There should be CRUD options available for problems in this view.
13. Add a new navigation header for all the user profiles - [183786253](#) (1 point):  
Create a custom navigation header to all the user profiles. When the admin logs in to code judge, a header specific to the admin role should be shown. Same will be the case for instructor and student
14. Score of a student - [183827694](#) (3 points)  
Once a student has uploaded the submission for a question, based on the number of test cases that were passed, the student should be able to see the score.

15. Access code for students to join a particular class just like piazza - [183827551](#) (3 points) :  
There should be an option in the student dashboard to join a class of his/her interest with a class code.
16. Option to upload instructor solutions and students shouldn't be able to see the solution. -[183827603](#) (3 points):  
There should be an optional way/option for instructors to upload their solution for a problem they create.
17. Tagging functionality on questions - [183400540](#) (3 points):  
In the questions tab, the instructor should be able to tag questions to one or more classes of his choice.
18. Renew the Glot API Token and update the backend code to evaluate the number of test cases passed - [183947263](#) (2 points)  
Currently the application is not able to connect to the Glot api, which is used to evaluate the student submission and sends back the result in json format. Modify the application code to show the test case evaluation time and to store the number of test cases passed.
19. Number of problems and questions in a class - [183828068](#) (2 points) :  
In the instructor dashboard, instructor should be able to view the count of students in the class and the count of questions in that class
20. Searching functionality on question tags - [183400534](#) (3 points) :  
Instructors and students should be able to filter questions based on tags.

## ***Completed bugs***

1. Unable to add problems to a class - [183807869](#) :  
With the existing functionality, instructors can only add a problem to a class with the help of problem id. Support should be given to add problems with the title. (A drop of all the available problems should appear to the instructor and support should be provided to select one of them)
2. Hide password in registration page - [183954799](#)
3. Question in instructor view should display question detail page on clicking the questions - [183954419](#)
4. Remove modify and delete from questions dashboard and insert them back in question details screen - [183954402](#)
5. Hide solutions for a problem in student view - [18395440](#)

## Completed story points

 Instructors should be able to che... <a href="#">view report</a>	20	20				Nov 18, 2022
 Solution of a problem <a href="#">view report</a>	9	9				Dec 2, 2022
 Class groups functionality <a href="#">view report</a>	18	18				Dec 4, 2022
 Final bug fixes <a href="#">view report</a> <i>No estimated stories</i>						Estimated Dec 11, 2022
 Future scope: Create group of qu... <a href="#">view report</a>						
 Statistics on class group <a href="#">view report</a>	7	2			5	
 Test case addition <a href="#">view report</a>	5				5	
 Test case visibility <a href="#">view report</a>	3				3	
 Sorting and searching functionalit... <a href="#">view report</a>	8	6			2	Estimated Dec 11, 2022
 Improving of existing code editor <a href="#">view report</a>	2				2	

There were three stories worth *seven points*, which didn't belong to any epic. Along with the points shown in the above burnup chart, The following stories were assigned different points (Refer below for specific distribution) each since they are big parts of the project and have all been completed.

1. Presentation : (2 points)  
This covered the slide deck of our work on codejudge.
2. Demo : (PPT demo + live demo - 3 points)  
This story covered the 6 and a half minute video demo walking through major use cases of our application
3. Report : (3 points)  
The current document which gives an overview of the project for a team that follows on from our application.

## ***Iceboxed user stories***

This section explains the user stories that were not prioritized or assigned points. These stories were created in the first iteration and were later ice boxed due to the fast-approaching deadline. This could be used as inspiration for additional features by the following team.

1. Improve the existing code editor for better code highlighting - [183400496](#)
2. Sorting of questions by tags - [183400533](#)
3. Implement test cases with different views - [183400556](#)
4. Implement batch upload test case - [183400628](#)
5. Improve existing functionality of test case upload - [183400616](#)
6. UI for group stats - [183400652](#)
7. Implement functionality on groups to perform statistics - [183400648](#)
8. Create group of questions for competition - [5047760](#)

## **Modifications to legacy project**

### ***Schema changes***

#### *Columns added*

- Passed column is added to the attempt table to store the state of the result. This may contain “failed”, “passed” or “grading in progress” values depending on the test case evaluation.
- Test\_score column is added to the attempts table to capture the score of a student in that attempt.
- Instructor\_solution column is added to the problems table to store the solution uploaded by the instructor.

#### *Tables created*

- We have created a new table ProblemGroups which contains information about mapping between the problem and groups table
- We have created a new table Tags which contains metadata about tags.
- We have created a new table ProblemTags which contains information about mapping between the problem and tags table
- We have created a new table StudentGroups which contains information about mapping between the student and groups table

#### *Tables modified*

- We have modified the groups table to contain all the metadata about groups.

### Relationships added
















- One to Many relationship between problem and problemGroups tables
- One to Many relationship between Groups and problemGroups tables
- One to Many relationship between Tags and ProblemTags tables
- One to Many relationship between Problem and ProblemTags tables

We went through each of the current requirements that we have, analyzed them and tried to understand if we could use the existing schemas as is. We have realized that though the existing schema is helpful to a certain extent, certain modifications are necessary in order to implement some of the new requirements, especially tagging and grouping. We made changes to the existing database, added new tables to them and ensured that none of the existing functionality is breaking because of the change in the table schemas.

### ***Migration changes***

Added new migrations to support new schema changes described above. Below is the list of new migrations that were added.

### Migrations added

 20221031024905_create_groups.rb	09-11-2022 11:55	Ruby File	1 KB
 20221031032511_create_problem_groups.rb	09-11-2022 11:55	Ruby File	1 KB
 20221031032752_create_tags.rb	09-11-2022 11:55	Ruby File	1 KB
 20221031032834_create_problem_tags.rb	09-11-2022 11:55	Ruby File	1 KB
 20221031032925_create_student_groups.rb	09-11-2022 11:55	Ruby File	1 KB
 20221031033219_adding_problem_group_foreign_keys.rb	09-11-2022 11:55	Ruby File	1 KB
 20221031033808_add_language_id_foreignkey_to_attempts.rb	09-11-2022 11:55	Ruby File	1 KB
 20221031034050_add_foreignkeys_to_problem_tag.rb	09-11-2022 11:55	Ruby File	1 KB
 20221031034241_add_foreignkeys_to_student_group.rb	09-11-2022 11:55	Ruby File	1 KB
 20221031034357_add_passed_column_to_attempts.rb	09-11-2022 11:55	Ruby File	1 KB
 20221101025738_rename_user_id_attempts.rb	09-11-2022 11:55	Ruby File	1 KB
 20221125031744_add_classcode_to_groups.rb	28-11-2022 13:32	Ruby File	1 KB
 20221125225020_add_test_score_to_attempts.rb	28-11-2022 13:32	Ruby File	1 KB
 20221126062236_add_instructor_solution_to_problem.rb	28-11-2022 13:32	Ruby File	1 KB
 20221126062237_add_problem_again_to_attempts.rb	28-11-2022 19:26	Ruby File	1 KB



### ***User profiling***

Changes were made to the registration page and the CodeJudge application to show the custom user interface screens to the user depending on their profile. Admin and Instructors will have a different header that supports their responsibilities respectively. Similarly, students have different components in their headers to support their functionalities.



## ***UI for different users***

We have made a lot of changes to the existing user interface of the CodeJudge application to enhance the user experience. These changes are the result of the new functionalities that were requested by the client. Some of them are:

- Students can join a class using a class code.
- Instructors are now able to group students into classes.
- Displaying the classes that the instructor has created
- Showing the questions that the instructor created

## **Roles of team members**

All the team members had played the roles of scrum master and product owner at least once. We followed a handover process, where the last product owner discussed the current sprint overview and the backlog tasks (which are ready to be picked up in the next sprint) with the next product owner. So it sets the tone and expectations for the upcoming sprint. Changes to the sprint stories will be made (if any) after meeting the client.

## **Iterations & Customer meetings**

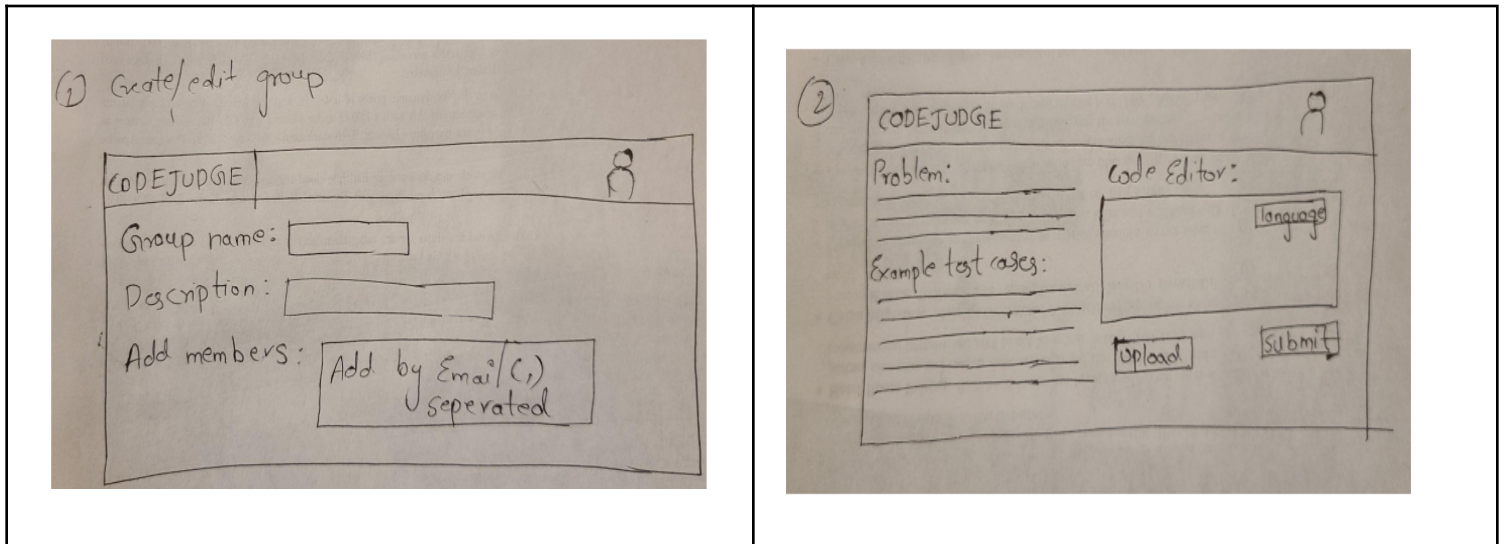
The following section describes the meeting summaries as well as a breakdown of what was completed in each iteration of the project.

### ***Iteration 0 + Meeting 0***

We met our client Dr. Ritchey on 09/23 for about 30 mins to discuss the requirements and to understand the CodeJudge application. Professor, described the requirement as a sort of an online coding platform like Kattis and Leetcode with some additional functionalities to them. In this meeting, the professor explained the deliverables. They are:

- Ability to add different kinds of tags to each problem.
- Sort and search functionalities by tags.
- Test Cases both visible and non visible[decided by instructor] to the user.
- Ability to add new test cases to problems by instructor both by clicking the add button and batch test cases addition(by a file upload).
- Ability to group students into classes and functionality to check statistics on a group.
- Instructors can create new groups and students can see/search available groups and
- Request to join.
- Improve existing editor by adding text highlighters for code.
- Instructor can check student progress.

Hence, Created eight user epics depending upon client requirements, each epic represents a feature which needs to be developed upon the existing code judge. We worked on some of the UI mockups. Some of them are:



## ***Iteration 1 + Meeting 1***

### **Completed Stories:**

- Install and Deploy the current code base - [183400443](#)

Everyone in the team focussed on setting up the local environment for development, So we cloned the existing git repository of code base to a local machine and started setting up the local development environment to run and finally deployed the application to Heroku Platform. As a part of the first iteration, since this is a legacy project our development team took the time to install the current version on their machines, set up the working environment and understand the code and the various parts associated with it. We had a group discussion, multiple times during this iteration where we all exchanged thoughts on how the application is working and understood different components in the code.

## ***Iteration 2***

### **Completed stories:**

- Schema changes for the database

### **In-Progress stories:**

- Add a new UI page for instructor view - (Mock up was completed)

- Add a new UI for admin view - (Mock up was completed)

After deploying the application locally and to Heroku last week, everyone on the team is ready to pick development stories. I have prioritized all the epics and identified the one that is a blocker for most of the remaining epics. That is "Instructors should be able to check and track the student's progress". Also, with reference to our first meeting with the client, one of our most important goals is to provide support for multiple profiles. We chose this epic because completing it will unlock other epics and potentially introduce new features to the Code Judge application.

Since it was the first time for the team to work on a ruby project, it took some time for us to understand the flow, syntax and other components of the application.

## ***Iteration 3 + Meeting 2***

### **Completed stories:**

- Add a new UI page for instructor view - (Spilled over from last sprint)
- Add a new UI for admin view – (Spilled over from last sprint)
- Navigate the user to their default home page depending on their user profile
- Modify Backend service to support recent UI changes

In this iteration, Team focussed on resolving the stories that were spilled from last sprint and successfully completed them. Since changes to the database schema were completed in the previous iteration, Hence, in this sprint we focussed on making changes to backend service to align with the new database changes. Also, utilizing the new database schema, we successfully completed the feature of navigating the user to their default home page depending on their role.

On 11/01 we met Professor Philip Ritchey and demonstrated new UI screens, new database changes and why the new schema is critical in delivering the user profiles feature. Based on the feedback from the client, we have created some other stories in the backlog that are necessary to deliver the feature.

## ***Iteration 4 + Meeting 3 + Meeting 4***

### **Completed stories:**

- Modify Backend service to support recent UI changes.
- Create a new view and UI for a class to support list of questions
- Instructor should be able to create classes or groups
- Instructor should be able to create or modify problems
- While registering, store the user type in database

- Create a new UI for a class to support list of students

In this iteration, the team focussed on the stories that are required to deliver MVP. We have focussed on delivering the critical functionality of the product, like creating classes, adding students to a class, adding questions to a class, only instructors should be able to create or modify problems and finally students should be able to see all the problems in all the classes he registered. We have also had a short demo to the professor on 11/15.

On 11/08 and 11/15, we met Professor Philip Ritchey and demonstrated the functionality we did. Professor Ritchey gave some feedback on the missing items and asked us to add some more new functionalities to the application. Some of them are: Students should be able to join a class with a code; students should be able to view the solution to a problem that instructor added to a class; and score should be visible to the student after he uploads the submission to the CodeJudge. Will be working on them in the next sprint.

## ***Iteration 5 + Meeting 5 + Meeting 6***

### **Completed stories:**

- Option to upload instructor solutions and students should be able to see the solution.
- Score of a student
- Access code for students to join a particular class just like piazza.
- Tagging functionality on questions
- Renew the Glot API token and update the backend code to evaluate the number of test cases passed
- Number of problems and questions in a class

### **Resolved bugs:**

- Unable to add problems to a class

In this iteration, the Team focussed on the stories that were requested by the client in our previous meeting. It includes several new features like creating an access code, calculating the score of a student, providing an option for the instructor to show the solution for students, tagging functionality on questions and a few bugs mentioned above.

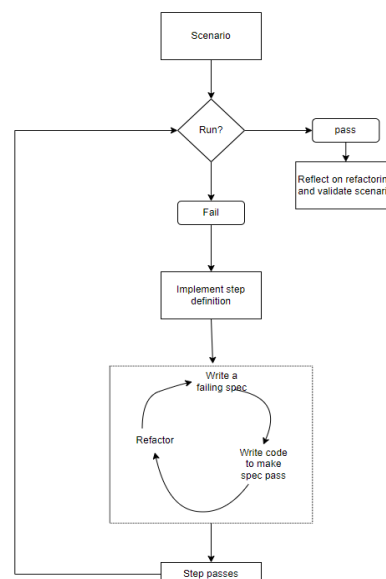
On 11/29, we met Professor Philip Ritchey and demonstrated the above features we accomplished in this sprint. Professor Ritchey was happy about the features and suggested a few modifications to the existing application. They are to write unit tests, improvise UI, and address a few bugs. Stories were created for those issues in the backlog and are currently being worked on.

Later we also met the Professor on 12/06, to demo the modifications he suggested in the previous meeting on 11/29. Professor Ritchey was satisfied with the changes and suggested documenting these bugs as well in the project report. Post that we had a discussion on project report structure.

## **BDD/TDD testing**

Behavior-driven development (BDD) is a software development practice of working in a short feedback loop, where we consistently apply test-driven development to every new feature we are exploring and working on. We have used RSpec as the main testing library and Cucumber for writing high-level acceptance tests. We have followed below steps for BDD cycle :

- We will start with one new Cucumber scenario.
- Run the above scenario and will wait till it fails
- Once the scenario has failed, we will write a definition of the first failing or pending spec.
- We will run the above scenario and will again wait till it fails.
- Test-drive the implementation of a Rails view using the red-green-refactor cycle with RSpec.
- RSpec should be used to test the controller using the red-green-refactor cycle. Check that the instance variables have been allocated and that the actions are responding appropriately.
- Use the same red-green-refactor cycle using RSpec to test those items. Check that they offer the methods required by the controller and view.
- Run the Cucumber scenario you started with once you have implemented all the objects and methods you require and the appropriate standards are passed to confirm that the phase has been satisfied.



# Git overview - Branches and Releases

For every new story or task, the respective developer creates a new branch from master. After completing the task, the developer commits all of the changes and pushes them to a remote branch. After the completion of testing, the developer raises the pull request to the master branch. Thereafter, some other developers will review the pull request and approve/reject the request. After successful review, the branch is merged to master. Below is the overview of the branches created in the git :

All branches				
LoginTypeIntegration	Updated 24 days ago by chanduakula-tamu	24   11	#9	Merged
questions-routing	Updated 24 days ago by ncchetan	24   12		New pull request
q	Updated 24 days ago by ncchetan	24   11		New pull request
feature/instructorView	Updated 24 days ago by rajendra1997	22   1		New pull request
instructor-crud	Updated 24 days ago by aravindwospace	24   10	#8	Closed
revert-6-instructor-crud	Updated 27 days ago by gokul-sai-doppalapudi	23   1	#7	Merged
iteration3	Updated last month by shyam prasad	26   1	#5	Merged
migrations_fix_sp	Updated last month by shyam prasad	26   1	#4	Merged
role-based-routing	Updated last month by ncchetan	27   5	#3	Merged
admin	Updated last month by chanduakula-tamu	28   1	#2	Merged
instructor	Updated last month by ncchetan	29   2	#1	Merged
origin/rajChanges	Updated last month by rajendra1997	29   1		New pull request

## Local Deployment

- As CodeJudge is in a Docker container, starting up the application is very simple. First, make sure Docker is installed and running on your machine. After cloning the repository, place the master.key file in the config directory. The below line should be placed in the master.key file

29c0db08284b68dd546254181ade0b6c

- After that, navigate to the codejudge directory and initialize the container by running the shell command

*docker-compose build*

- Make sure you have installed the postgres and redis in your local machine.
- If this command returns an error, make sure the docker is running. This step may take a while if it's your first time building the container, but in later iterations Docker caches many of the components it needs. Once the container is built, run the shell command.  
*docker-compose run web rails db:create db:schema:load db:seed*
- This command initializes the database, loads the schema from schema.rb, then seeds the database with fake data as specified in seed.rb. Once the database is built and seeded, run the below shell command.  
*docker-compose up*
- If you want to recreate the container from scratch later, run the below shell command.  
*docker-compose down -v*
- Environment variables are handled with the Figaro gem. You'll need to set your environment variables in the application.yml file. You will need to set your domain as an environment variable as well as your glot.io API key. Other variables are:
- *RAILS\_MASTER\_KEY* & *SECRET\_KEY\_BASE*: have to be set in Heroku under Settings→Config Vars
- *RACK\_ENV* & *RAILS\_ENV*: set to production
- *RAILS\_SERVE\_STATIC\_FILES*: set to enabled
- *GLOT\_KEY*: get by registering for the glot.io API
- *DOMAIN*: your app domain, e.g. appname.herokuapp.com
- *REDIS\_URL\_SIDEKIQ*: has to be the same as the *REDIS\_URL* that is set as soon as you add the Redis Add-On to Heroku
- *DISABLE\_DATABASE\_ENVIRONMENT\_CHECK*: set to 1

## Heroku release process

### *Overview*

- First, download and install the [Heroku CLI](#).
- Run the command  
*heroku login*
- Once you're logged in, you can log into the Heroku container registry with  
*heroku container:login*
- To deploy your container, you must be in the directory with your Docker image. If it is the first time deploying to heroku you have to add the Heroku Postgres and Heroku Redis Add-On first. To push the container to Heroku, run  
*heroku container:push web sidekiq --recursive*

- Then, to deploy your changes to the production site, run  
*heroku container:release web sidekiq*
- Again, if it is the first time deploying, the *sidekiq* dyno should now show up under the Resources tab on Heroku. If so, go ahead and enable it.
- Since the Heroku database is the production database you do not have to create it first. All you have to do after you push the container is running migrations on the production database. If it is the first time, do  
*heroku run rails db:schema:load db:seed*
- For future deployment, just do  
*heroku run rails db:migrate*  
and you're done!

## ***Issues faced***

- Sometimes, we need to downgrade the stack version of heroku to make the application run. This error is observed when we are deploying the code base to heroku. To resolve the error / issue, we used to run the below command which sets the stack to version 20.  
*heroku stack:set heroku-20*
- The Git repository for CodeJudge and Heroku are different, In specific heroku only requires the root directory of CodeJudge. So every time it's a hassle initializing the git repository to deploy the latest version to heroku.
- Sometimes we may get an inconsistency error between heroku remote repository and your local repository. To resolve this, you need to either pull the heroku remote repository and push your changes OR do a git force push.

## **Software tools used**

### ***Overview***

We have used a lot of open source tools which helped us to efficiently understand and resolve the errors/issues that we were facing. Below is the list of tools used in the development process :

- Datagrip by jet brains : It is a new database IDE that is tailored to suit the specific needs of SQL developers. (Used by almost all of us to interact with database)
- Redis CLI : The Redis command line interface (redis-cli) is a terminal program used to send commands to and read replies from the Redis server (used for testing and starting sidekiq)
- Postman : Postman is an API platform for developers to design, build, test and iterate their APIs ( Used for testing Glot API, which evaluates the student's submission)



## ***Issues***

As the above sources were widely used and have high compatibility with development environments. We haven't had any issues with these tools.

## ***Benefits***

- To visualize the data in our production environment (heroku), Datagrip was very handy and it made debugging very easy for issues related to databases (constraints violation, duplicate data, etc).
- Postman is also a great tool for debugging API's. It supports all types of HTTP requests with a customized support to various authentication mechanisms. We have created a new token in the Glot API and tested some of the GET and POST requests to evaluate a code with the new token.

## **Testing - cucumber and Rspec**

Testing is performed using regular RSpec as well as Cucumber, a Ruby on Rails testing framework that interacts with the browser in the same way that a user would. We've developed a couple of Cucumber feature files and several step definitions so far. The cucumber tests may be performed using the following command:

```
docker-compose run web cucumber
```

We have added a SimpleCov metric to our repository, however we have faced so many errors due to the docker container. We have tried to understand and resolve the error, but we couldn't. We have used this [source](#) as reference to understand SimpleCov. To get results, we followed this process: execute the above command, then download and run the codecov uploader utility on your local PC.

## **Git repo contents**

The Git Repository mainly consists of 2 directories and a file: codejudge - that has the project codebase files, documentation - that contains all the project documentation and iteration reports and a README file - that briefly describes about the project and the steps to setup the project locally and deploy to Heroku. After setting up the project in the local machine, *cable.yml* and *database.yml* files that are present in the config directory of the codejudge directory, can be configured to set up the local database. We made sure that all required files such as *docker-compose.yml*, *heroku.yml*, *Dockerfile.sidekiq* to deploy the project are available in the git repository.

## **Project links**

- [Pivotal tracker](#)
- [Heroku deployment](#)
- [Github Repository](#)
- [Presentation and Demo video](#)