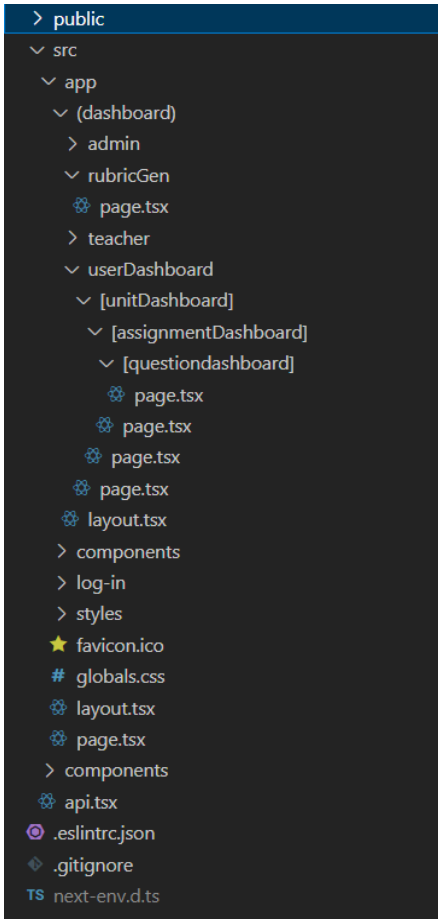
Developer Manual - Frontend

Assumptions:

- The developer reading the manual is familiar with the next.js framework and file structure.
- Directories or files ignored in the manual are redundant or are unmodified since the default generation of the repository by the framework itself.
- All components are called from “src/app/components/”
- User emails are the keys used to fetch data from the backend and are hardcoded into every request due to not having authentication done
- Sample files and manuals are found in “@/manuals and sample files/”
- When testing the repository
    - StudentList CSV files must be in the format specified in “StudentList.csv”
    - Random questions in QuestionTemplate upload must be in the format specified by “dummy_questions.csv”
    - Marking Guides for conversion to rubric must be in the format specified by “Sample_MarkingGuide.pdf”



The above image shows the file structure of the Frontend repository.

The “/public” directory holds all of the `.png` files used in the application.

The `api.tsx` file in “/src” holds all the constants used to make API calls to the backend. It also has a constant variable called `BASE_URL` where the IP of the backend can be changed and will take effect in the whole application.

“/src/app” is where the vast majority of the application's code is. “src/app/styles” holds all of the `.css` files used to style the respectively named components and files. “src/app/page.tsx” is the first page that is rendered on the application. It shows the Viva MQ logo and calls the `SignInForm` component which is a simple boilerplate authentication form that does not work.

“src/app/(dashboard)” is the base route of the application and has components on it that will be rendered on every single page. Inside of “/(dashboard)”, there is “/userDashboard” and “/rubricGen”.

UserDashboard

“src/app/(dashboard)/userDashboard” contains a `page.tsx` file which the code for the main dashboard the user will interact with is stored. As well as this, inside there is a `[unitDashboard]` which stores the nested code of when you actually create a unit. Inside the `page.tsx` file of “/userDashboard” there are multiple moving parts that come together to have the complete functionality. The first is the `UnitCard` component, which is populated by GET requests from the backend for a particular unit. These components are mapped so that all available units are shown. There is also the `CreateUnitPopup` component being called which is a form that takes the relevant data in from the user to create a unit. Once a unit is displayed, the user will click on it to route it to the `[unitDashbaord]`.

The nested structure of “/userDashboard” containing `[unitDashboard]` extends and so `[unitDashboard]` contains `[assignmentDashboard` and also contains its own `page.tsx` file. In this `page.tsx` file, the code is responsible for the user view inside a particular unit, as well as the ability to CRUD TAs onto a unit, and upload the student list. The CRUD of TAs functionality comes from the `TutorList` component, and the uploading a student list functionality comes from `UploadModal`. The reason `UploadModal` is its own component is because it is a popout component, similar to the `CreateUnitPopup` component. Once an assignment is created, it will be mapped and rendered using the `Assignment` component, which also houses the edit and delete functionalities for that specific assignment. Once an assignment is created and displayed, it can be pressed to trigger the code in `[assignmentDashboard]/page.tsx`.

Inside of `[assignmentDashboard]/page.tsx`, the code is responsible for what the user sees when they are required to upload the question template, the student submissions and select what files they want a viva generated for. All the code responsible for all this functionality is found within this file, except for the popout functionality of the question template form which is found within the `QuestionTemplate` component. Once a user has gone through the process of generating a viva for the student submission, they can click on the title of the submission to execute the code that is in the `[questionDashboard]/page.tsx` file.

Inside of the `[questionDashboard]/page.tsx` is very simple. It purely passes the correct params to a `ReviewQuestions` component, which houses the interfaces that display the static questions passed in the question template, the random questions uploaded inside of the questions template, and the functionality of regenerating specific AI-generated questions the user does not like.

RubricGen

In the “/rubricGen” directory, there is a `page.tsx` file. This file contains the code used specifically for the retrieval of rubrics from the backend as well as displaying and formatting them. It also houses the buttons used for `.PDF` and `.XLS` file export for each rubric. Three key components that are called within this file are: `ConvertMarkingGuidePopup.tsx`, `CreateRubricPopup.tsx` and `EditRubricPopup.tsx` which are all forms that popup over the “/rubricGen” page when their respective button is pressed.

Inside of `ConvertMarkingGuidePopup.tx` the structure is similar to all other forms found within the application. The major difference with this file is that on the `.PDF` of the marking guide, it will use the returned ID from the POST request to populate a hidden field in the form, then will submit it with the ULOs and the hardcoded convener email. Once submitted, it will show “Submitting” on the button until the form closes, and then the main page of “/rubricGen” will be seen with the newly generated rubric.

Inside of `CreateRubricPopup.tsx`, it is quite similar to the rest of the forms inside the application. This form specifically has separate text box fields with their own remove buttons for their own respective fields as in the code it makes it easier to format the JSON file in the way the backend expects it. This removes the need for string manipulation at the user's UX. Once submitted, it will show “Submitting” on the button until the form closes, and then the main page of “/rubricGen” will be seen with the newly generated rubric. `EditRubricPopup.tsx` is the exact same form as `CreateRubricPopup.tsx` however it’s fields are prepopulated based on the respective rubric it is editing.

User Manual - Frontend

Assumptions:

- Do not upload excessively large-sized files when uploading student submissions
- Do not use any “emojis” when filling out forms
- When testing the repository
    - StudentList CSV files must be in the format specified in “StudentList.csv”
    - Random questions in QuestionTemplate upload must be in the format specified by “dummy_questions.csv”
    - Marking Guides for conversion to rubric must be in the format specified by “Sample_MarkingGuide.pdf”
- Deployed URL is: https://comp-4050-churros.vercel.app/

The first page you will land on is the Authentication page ("/"), as of right now doesn't actually work however if you click to submit it will redirect you to a logged-in perspective of a user dashboard. The underlying user that you will be authenticated as is “convener1@example.com”. All functionality going forward will work the same as if authentication was implemented.

Viva Generation:

Now on the ("/userDashboard") page, you will be able to create a unit that you want to generate Viva assessments for. Once you click the create unit button and fill out the form, the unit you created will come up and you will be able to now click on it on your dashboard.

You will then be redirected to ("/userDashboard/unitName") where you can create, edit, view and delete the assignments for that particular unit, or you can add, edit, view and delete the teaching staff assigned to that unit. There is a search feature on the teaching staff lookup to make it easier for you. Please note on this page, you are expected to upload a `.CSV` file that has the student list of that unit in the pre-agreed-upon format. Once you have created an assignment and click it you will be redirected to ("/userDashboard/unitName/assignmentName"). On this page, you will be expected to first upload a question template, which will mould the questions you want generated for the viva voce.

Static questions in the question template refer to the constant questions that will appear in every viva. Next, the random questions will be a CSV with a list of questions in the pre-agreed-upon format. Finally, you will enter how many of each AI-generated question you would like. Once this form is submitted, you will upload the student submissions. You can do this in a batch file upload rather than individually. Once you upload the student submissions, a list of the submissions should appear (if they do not, refresh the page).

You can now select the box next to each submission or you can select the top left box on the table to select all. Once the boxes are ticked you can click the generate questions button. After approximately 5 seconds, the status of the submission will change from “Generating” to “Uploaded”, just ignore this; once the status is “GENERATED” you may now click the title of the submission and it will route you to the viva page of the submission. Inside you will see the tabs, static, random, and AI-Generated. The static and random are view-only tabs, whereas the

AI-Generated tab will allow for you to regenerate specific questions you did not like. The process for this is to click the box next to it, click the reason for regeneration and select regenerate. You can select multiple at a time.

Rubric Generation

Now from the ("/userDashboard") page, or anywhere in the application, you can use the toolbar on the left to click “Rubric Generation” which will route you to ("/rubricGen"). On this page, you will by default see all the rubrics currently generated for the aforementioned user. If you click the “Create Rubric” button, a form will pop up for you to fill out. Once you submit this form, the submit button will change from “Submit” to “Submitting” and eventually the form will close. At this point, you can refresh the page if it does not automatically and the rubric you requested to generate will appear.

If you click the “Convert Marking Guide Button” a small form will appear, in this form you must first put in the marking guide in the pre-agreed-upon PDF format, then you can fill out the form and submit it. The submit button will change from “Submit” to “Submitting” and eventually the form will close. At this point, you can refresh the page if it does not automatically and the rubric you requested to generate will appear.

If you click the view details on any rubric review, a list of options will appear alongside the actual rubric itself. The edit rubric button will prompt a form where you can edit the rubric, the delete rubric button will remove the rubric, and the two export options will download in the respective format you pick.