Two paragraph summary of the project *as implemented*, including the main customer need and how the application meets it, including who the stakeholders are. This will contrast to what you wrote in Iteration 0.

//TODO

Description of *all* user stories (including revised/refactored stories in the case of legacy projects). For each story, explain how many points you gave it, explain the implementation status, including those that did not get implemented. Discuss changes to each story as they went. Show lo-fi UI mockups/storyboards you created and then the corresponding screen shots, as needed to explain to stories.

**User Stories**

**Feature: Log in to the system**  
As the main admin,  
So that I can access the system with read/write privileges  
I want to securely login to the application

Story Points: 2

Status: Completed

The clients wanted a simple email-id and password login system since the application will be deployed in a private VPN anyways.

Graphical user interface, application, website

Description automatically generated

**Feature: View the data in tabular format**  
As the general/main admin  
So that I can keep track of all the records  
I want to view the records in an organized tabular format

Story Points: 4

Status: Completed

This story involved receiving the data from the backend and displaying it in a tabular format with relevant columns for the client shown in front.

Graphical user interface

Description automatically generated with medium confidence

**Feature: Super search the data**   
As the general/main admin,  
So that I can find specific entries by word  
I want to search for students, courses, etc.

Story Points: 6

Status: Completed

This story involved creating a common search field that can traverse all columns in one search.

Examples:

Searching by Company

Graphical user interface, text, application

Description automatically generated

Searching by Name

Graphical user interface, text, application, email

Description automatically generated

Searching by email

Graphical user interface, application

Description automatically generated

**Feature: Upload data using csv**   
As the main admin,  
So that I can view/search/filter data I want to store the data using csv

Story Points: 17 (3 – frontend, 4 – CSV type 1, 4 – CSV type 2, 3 – adding an upload modal, 3 bug fixes)

Status: Completed

This story involved taking any CSV that client provides and respectively adding data into the database and showing in the frontend. This user story was challenging because the CSVs had no one format, and thus there are many different keys and types of data.   
Example: One CSV might have “Name”, while the other would have “First Name and Last Name”. So, for the former CSV, a logic had to be written to separate into first and last name. Similarly, all other keys had differing formats.

**Feature: Add data through forms**   
As the main admin,  
So that I can enter the user data through a form-like interface I want to manually enter the respective data

Story Points: 24 (5-CRUD APIs, 15 - Form creations, 4 - Form validations)

Status: Completed

Modification: This story was modified later. Initially, a form was to be created to only add a student but further it was extended to have a form for all different fields namely student, course, company, certificate vouchers, vendors and users.

This story involved a lot of effort since different forms had to be created for all different fields. They have separate keys, formats, APIs, and even data validation techniques.

Example:

Graphical user interface, website

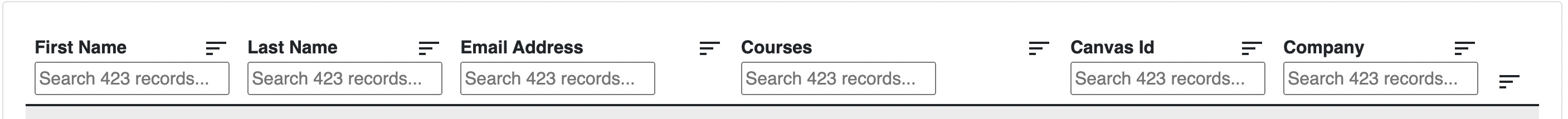
Description automatically generated

**Feature: Filter student records by a column**   
As the general/main admin,  
So that I can find specific entry by a column field I want **to** filter students by a column

Story points: 7  
Status: Completed

Modification: This story was modified to filter on all columns rather than the earlier version where it can filter only on company name.

In the image below, we can see all columns having their own respective search fields.



**Feature: View, Edit student profile**   
As a general/main admin,  
So that I can get all the information on a specific student I want to view or edit the student’s profile

Story points: 10

Status: Completed

Modification: Earlier the student profile was supposed to be read-only, but later we added edit functionality as well.

The student profile here shows all information relevant to a student, and option to edit them. Also courses information can be added and viewed as well.

**Graphical user interface, text, application

Description automatically generated**

**Feature: Navbar to access different components (Students, Exams, Vendor, Users, Company, Courses, Certificates**  
As the general/main admin,  
So that I can access or view the information that I am looking for   
I want to find the detailed information about each component

Story points: 5

Status: Completed

Modification: This story was not initially planned but decided upon after the 1st demo with the client

In this story we created different navigation components related to different type of data fields we have.

Graphical user interface, application

Description automatically generated

**Feature: Export data**  
As the main admin,  
So that I can view, save, and share data  
I want to export data into csv format

Story points: 6

Status: Completed

Modification: This story was not initially planned but decided upon after the 1st demo with the client

The clients wanted an export to CSV feature where when they are filtering or looking up any data records, they can easily export them to a CSV for retention or sharing.

**Feature: Edit and delete records**  
As the main admin,  
So that I can edit and delete data  
I want to have options on the respective tables

Story points: 5

Status: Completed

Modification: This story was not initially planned but decided upon after the 1st demo with the client.

The client wanted a quick edit and delete option on different tables we have for different fields. It looks like this

Graphical user interface, text, application, email

Description automatically generated

**Scrum master and product owner**

We changed the scrum master and product owner every iteration. Following was the order of roles assumed:

|  |  |  |
| --- | --- | --- |
| **Iteration** | **Scrum Master** | **Product Owner** |
| 0 | Yatin Gupta | Mesbaul Alam Khan |
| 1 | Sabina Adhikari | Aayush Gautam |
| 2 | Anuska Pant | Yatin Gupta |
| 3 | Mesbaul Alam Khan | Sabina Adhikari |
| 4 | Aayush Gautam | Anuska Pant |
| 5 | Yatin Gupta | Mesbaul Alam Khan |

For each scrum iteration, summarize what was accomplished and points completed.

// TODO

**Customer Meetings:**

1. Initial Meeting: 20th September 2022

In this meeting, we discussed the requirements of the client. The client showed us the rudimentary management system they have using a lot of excel sheets, and what they wanted to have. We went over all their data, and what exactly they are looking for in the application. Most of the user stories were defined in this meeting, the approach to database schema was discussed and a big overview of the application was formed.

1. ERD discussion: 27th September

Based on the initial meeting, we developed an ERD schema and went back to the client to discuss the edge cases and come to a consensus on how the database should look. Since the client has a lot of different data points we wanted to make sure we are aligned before work is started on designing the database schema.

1. 1st Demo: 21st October

All the earlier discussions led to this moment where we came prepared with a proper V1 of the application developed. We showed the client the deployed application on Heroku going through the various features we had. Particularly, we showed the following user stories:

1. Log in to the system
2. View the data in tabular format
3. Upload data using csv
4. Super search the data

The client was very impressed with the product, and it covered all the major features they wanted. Next up, we presented some of the future ideas we had to make the application better and client also told us some of the other features or improvements they wanted.

1. 2nd Demo: 16th November

In this meeting, we showed all the features and improvements that they had asked after our last meeting. Particularly, we showed the following user stories:

1. Add student data
2. View student profile
3. Filter student records by company name
4. Navbar to access different components (Students, Exams, Vendor, Users, Company, Courses, Certificates

The client was happy with the application. But later we noticed an inconsistency in the database schema due to miscommunication between the team and the client. It was a small error in the schema where the relationship between “exam” and “voucher” was misunderstood. We took note of this and suggested some design improvements to the client that we wanted to take up. The client also discussed with us V2 version of the application that they will work on in the next semester.

Afterward, we implemented all the changes, design improvements, and pending user stories. Due to the busy schedule of our clients, we have been communicating through email to show the finalized product.

**BDD/TDD process**

We followed a pairing process to do BDD/TDD. Instead of one person writing tests as well as the feature together, in our approach one would write the BDD test (cucumber), one would write the system tests (rspec) and finally other would do the feature development.

Benefits:

1. This way we removed the biases or blind spots a person may develop for a feature.
2. This also worked in pair programming fashion where 2-3 people are working on 1 feature-related code.
3. Code review became easier since 2-3 people had context on the entirety of the feature

Problem:

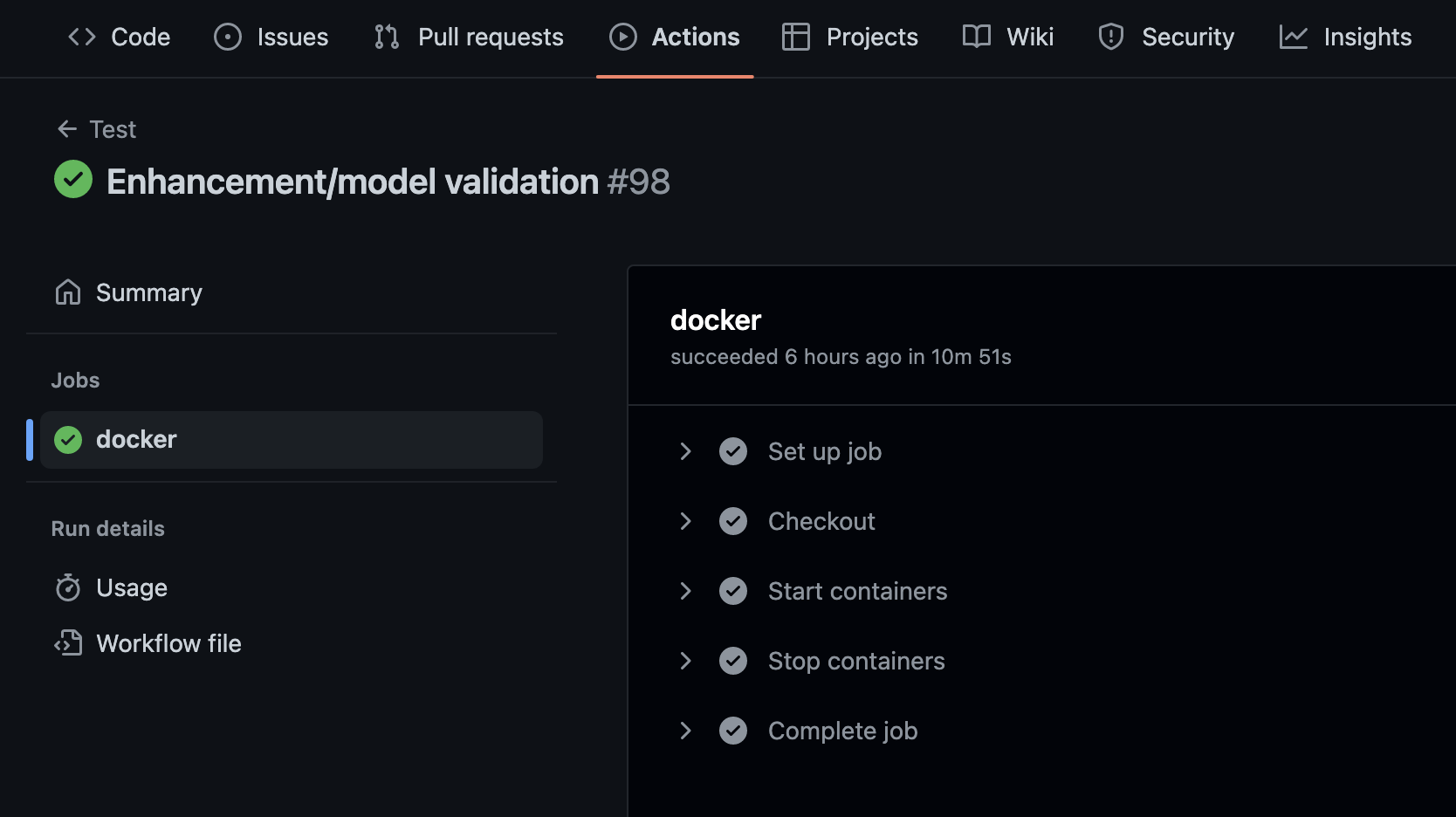
1. Different understandings of same feature caused delays since tests and features were developed by different people
2. Sometimes bad code was pushed since tests were being developed in a separate place

Although our approach had its benefits, the traditional TDD approach where tests are being written with the feature seems much better to avoid bad code causing issues.

**configuration and release management**

We followed the following steps for managing releases:

1. All work was done in a separate feature/bug branch.
2. All Pull requests went through automatic **CI process** using github actions. Our Rspec and Cucumber tests would run in a docker container inside a runner provided by github actions.
3. A Merge request was raised to main which required atleast one approval before it can be merged.
4. If the tests fails or there are any review comments, the developer fixes those
5. If the CI pipeline successfully passes and approver is happy with the merge request, it is merged onto main branch
6. We did 1-2 releases per iteration depending on the changes. We tried to follow fast releases and verify the changes on our production URL.



To be updated on notifications of all merge requests, we integrated slack with GitHub.

Text

Description automatically generated

**Production Release**

There is an issue in the release to Heroku due to tool versions. We are using ruby version 2.7.2 and rails 6.0.6 version. Due to Heroku not supporting this version, we had to use Heroku 18 which has its end of life as April 30th, 2023.

This is not an issue with our client since the client wants to deploy the application in their own VPN. To help expedite the process for them, we **dockerized** the entire application so that the client can easily deploy it as a container.

**SimpleCov**

To find out the code coverage for our tests, we used SimpleCov. It is a simple tool that succinctly integrates with rspec and cucumber. We only need to run both tests once; it merges the code coverage and gives us the final report. It shows the final code coverage in % as well as the lines we covered or missed. This was useful in identifying features/corner cases we forgot to test.

Graphical user interface, application, Word

Description automatically generated

Here the green line shows the successfully tested lines while the red ones show untested lines. Graphical user interface, text, application

Description automatically generated

**Repo contents**

We used the ruby on rails monolithic approach to design the application. The backend is rails, while in the frontend we replaced normal JavaScript or templating with react in the monolith**.** Standard rails folder structure is followed with some modifications/additions.

* **.github** folder has the yml file needed to run tests with github actions.
* App represents all the business logic code needed
* Frontend is in react, and code is present in app/javascript
* Documentation has respective tar files done for each iteration
* Features folder has the cucumber tests

For deploying to Heroku, it is just connecting with the project and running git push heroku main. But since our clients wanted to deploy the application in their own VPN, to help speed up the process we dockerized the entire application and added respective docker and docker-compose files in the repo.

**Links to your Pivotal Tracker, GitHub repo, and Heroku deployment. Make sure these are up to date.**

**Pivotal Tracker**: https://www.pivotaltracker.com/n/projects/2598723

**GitHub repo**: https://github.com/ANUSKAPANT/CyberSec-Cert-Tracker

**Heroku**: <https://cybersec-cert-tracker.herokuapp.com/dashboard>

For testing: use these credentials

**Users**

Email: [test@example.com](mailto:test@example.com)

Password: test123

**Admin**

Email: [admin@example.com](mailto:admin@example.com)

Password: test123

Links to your presentation video and demo video.

//TODO