FUDGE E-INVOICING

Requirements and Design Workshop
SENG2021

Final Design Report

18 April 2022

Chantelle Conlon Scoullar – Scrum master / lead backend developer
Gio Ko - lead frontend developer
Aishani Mathur - backend developer
Sanjana Dinesh – delivery manager /backend developer
Joel Abraham – product manager / backend developer

Class W14A

Contents

1.0 Introduction

1.1 Abstract

This report explores the designing of the e-invoicing system by Fudge Invoices inc. The report begins by visiting the key points of how requirements and constraints were considered when designing the program, whilst also creating specific use cases for different user stories. This report will also examine the software architecture and how the different layers were essential in creating the final product that has been deployed by Fudge Invoices Inc. The final API design as well as the lifecycle of the API being used through different stages of creating the product is further discussed within the report.

1.2 Team Roles

Fudge Invoices Inc consisted of a cross functional team with both backend and frontend developers dedicated to producing a quality e-invoicing system.

Team roles were allocated on a nomination system. Our team understands that team roles not only distribute responsibility, they also help streamline the production of our product.

- **Scrum Master,** Aishani Mathur – As a natural leader, Aishani was able to utilise agile project management techniques to ensure success and delivery of the e-invoicing platform, working hand in hand with all team members.

- **Product Owner**, Joel Abraham – With an invested understanding into the required functionality of the platform, Joel was able to lead the requirements gathering, and use case design to streamline execution whilst maintaining conceptual integrity of product features.

- **Delivery Manager**, Sanjana Dinesh – With good communication skills and helping the team keep track of what's coming up and what to prioritise first, Sanjana proved that she was ideal for this role by utilising team resources and activities to create the final product.

- **Senior Front End Developer**, Gio Ko – With a significant understanding of JavaScript and methods on how to develop a frontend website, Gio was well position to own and drive UX and UI efforts.

- **Senior Back End Developer**, Chantelle Conlon Scoullar – With a deep understanding of backend design and server creation, Chantelle was excited to take point on leading the complex logic design.

2.0 Requirements Engineering and Constraints Analysis

2.1 Requirements Engineering

Features

- Receive an uploaded e-invoice
- Receive an e-invoice sent by another business
- Receive an e-invoice emailed by another business
- Generate a communication report on details of received invoice
- Search for a communication report
- Search for Invoice
- Generate a rendered downloadable pdf of invoice

Functional Requirements

1. As a small/medium-sized enterprise owner when I upload an e-invoice, I want the system to receive the invoice and generate a communication report on the details of received invoice.
2. As a small/medium-sized enterprise owner when I am sent a e-invoice via email or the through the system, I want the system to receive the invoice and generate a communication report on the details of received invoice.
3. As a small/medium-sized enterprise owner when I search for a particular received e-invoice, I want the system to generate a communication report on the details of received invoice
4. As a small/medium-sized enterprise owner when I search for a particular invoice, I want the system to generate a rendered pdf of the invoice.

Non-Functional Requirements

1. Validity
   As a small/medium-sized enterprise owner, I want correct security measures to be that assures consistently correct content in the invoices.
   Given that I am receiving an invoice as a UBL XML file then I should be getting the correct information sent.
   - Given that I am receiving an invoice as a UBL XML file then when the invoice is being translated into json, json or html the data should match as the invoice I received.
2. Security
   As a business owner, I want to be able to protect the access to my invoices.
   - Given that I want to access my invoices and open the application, I want to be prompted to log in system.
   - Given that the software prompts a login, only individuals who have managerial authority should be able to access the invoices.
3. Reliability
   As a small/medium-sized enterprise owner, I want a fast and reliable service to receive invoices
   - Given that I am receiving an invoice, then it should take no longer than a minute for the invoice to be successfully translated to a readable format.
4. Accountability

As a small/medium-sized enterprise owner I want my invoices to be securely stored.

- Given that I received an invoice, I want the system to have a log of the date and time it was received.
- Given that I want to access details about how the received invoices were manipulated, the system should also have a log of all these details. Note: Manipulations refers to opening, closing, deleting invoices.

Use Cases

1. Given an SME conducts business with another company, when they complete their transaction, then they want to automatically receive their e-invoice through an online platform.

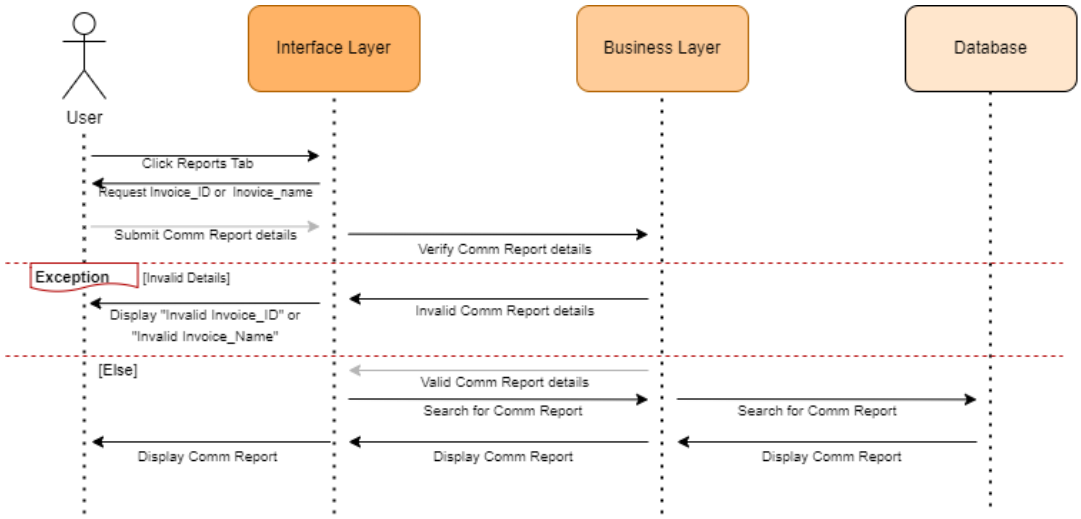


Figure 1: {ADD DESCRTIPTION}

2. Given that the adoption of UBL is becoming the preferred standard message representation when a business receives an e-invoice, they should be assured that all invoices are received in the correct format.

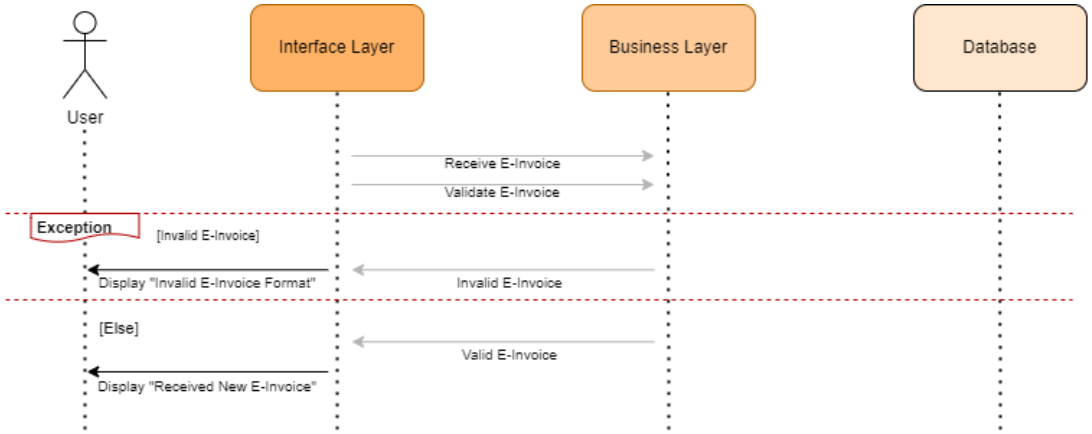


Figure 2: {ADD CAPTION}

3. When a business owner receives an xml e-invoice file they want it to be automatically converted into a human readable format when they display its content.

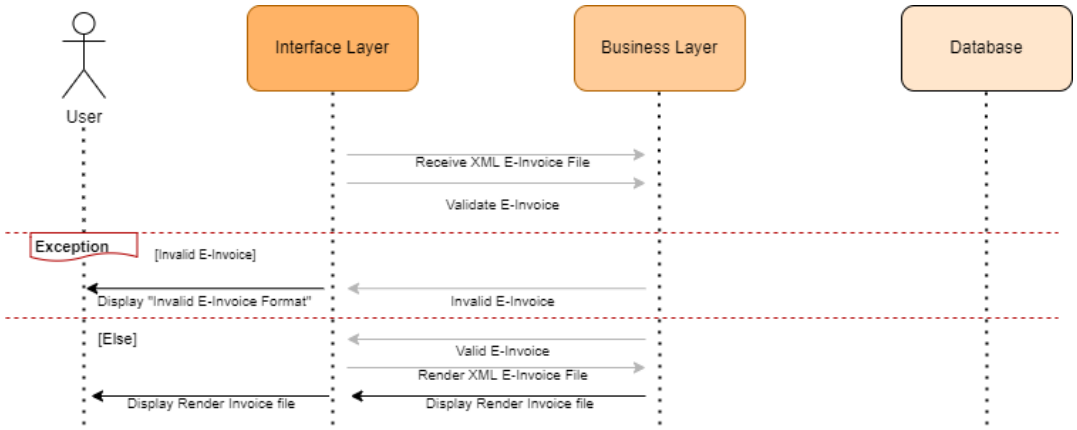


Figure 3: {ADD DESCRIPTION}

2.2 Constraint Analysis

Table 1: Constraints analysis

Constraint	Measures to be taken to exploit constraints
Limited amount of knowledge we currently obtain, compared to the skills/knowledge we will need to competently complete the project.	• All team members will conduct research in areas they feel like they need to brush up or add to their knowledge. • We have also planned regular team meetings where we discuss and resolve issues people are facing together as a team.
Limited time we are given to complete our project.	• Creating a thorough project plan in accordance with all our due dates for every sprint. This will help us ensure we stay on track and don't waste too much time on a single task requiring us to rush our work towards the end of project deadlines.

3.0 Software Architecture Design

3.1 Software Stack

The software architecture can be broken down into 3 layers being the deployment layer, interface layer, and services layer.

3.1.1 Deployment Layer

The deployment layer presents users access to the product from anywhere on the internet. This simple way to access the API is enabled since it would be running on an online server.

By having this deployment layer, our product allows small-medium enterprises and individuals, who want to effortlessly understand their invoice, to use our product if they have internet connection. We used the free version of the Amazon Web Services (AWS) platform to deploy our service. After extensive research into other deployment services such as Heroku and AlwaysData, it was evident that this platform would be the most suitable for our product. We trialled Heroku, however, we were receiving several token errors and it was slow, taking up to a minute to load our application. It also had git integration issues that rendered it difficult for us to deploy via our GitHub. AWS had features we could use as we upscaled our invoice receiving API such as storage functionalities called S3.

3.1.2 Interface Layer

The interface layer is essentially the user interface and will act as a shield from all the services or APIs being used from the user. We used JavaScript, HTML, and CSS to develop our interface layer. JavaScript was chosen as it can easily be loaded from HTML files to power the frontend. HTML and CSS was chosen to structure our webpage and content with an aesthetic and clean design.

To navigate through the interface, use the following link: http://invoiceplatform-env.eba-9xzpwpa2.us-east-1.elasticbeanstalk.com/.

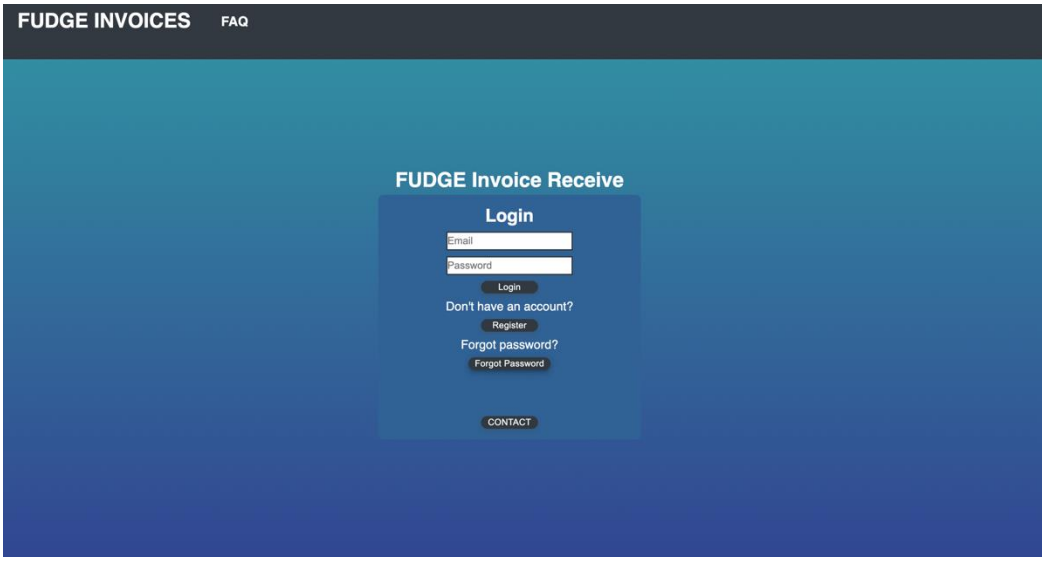


Figure 4: Landing page of website.

The user is initially greeted with the page necessary for authentication as shown in figure 4. The interface layer displays a user-friendly frontend that provides key information that is required for cohesive usage of the product. Backend code that would, for instance, test for a valid authentication is translated into the frontend in the form of a prompt from the system (as shown in figure 5). For e.g. If a user attempts to login prior to registering, the prompt “Wrong email or password” is displayed to the user, along with options to register.

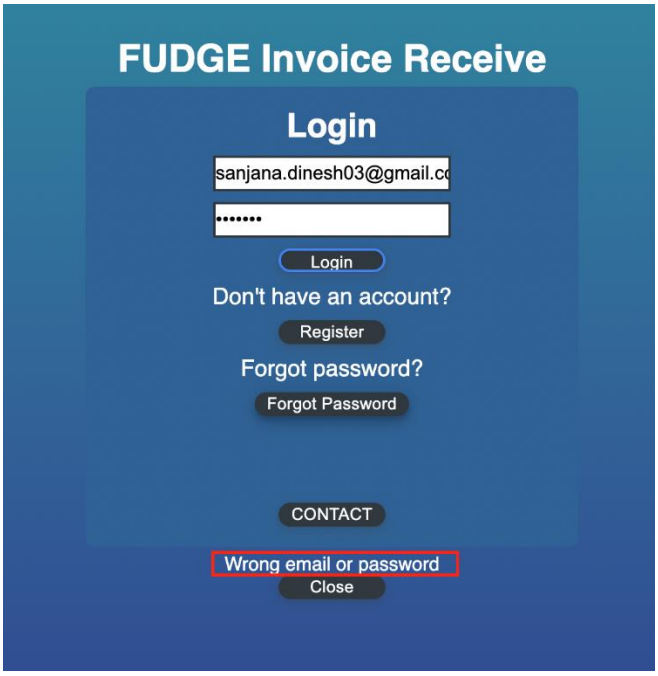


Figure 5: System communicates to user via prompts.

Another example of the user interface would be when the system provides the user feedback and information about the xml file they submitted. This informs the user that they gave the correct file for the system to receive (shown in figure 6). In cases, where the user attempts to upload an incorrectly formatted xml file, they receive a pop up that tells them that it is incorrect and are stopped from uploading it (shown in figure 7).

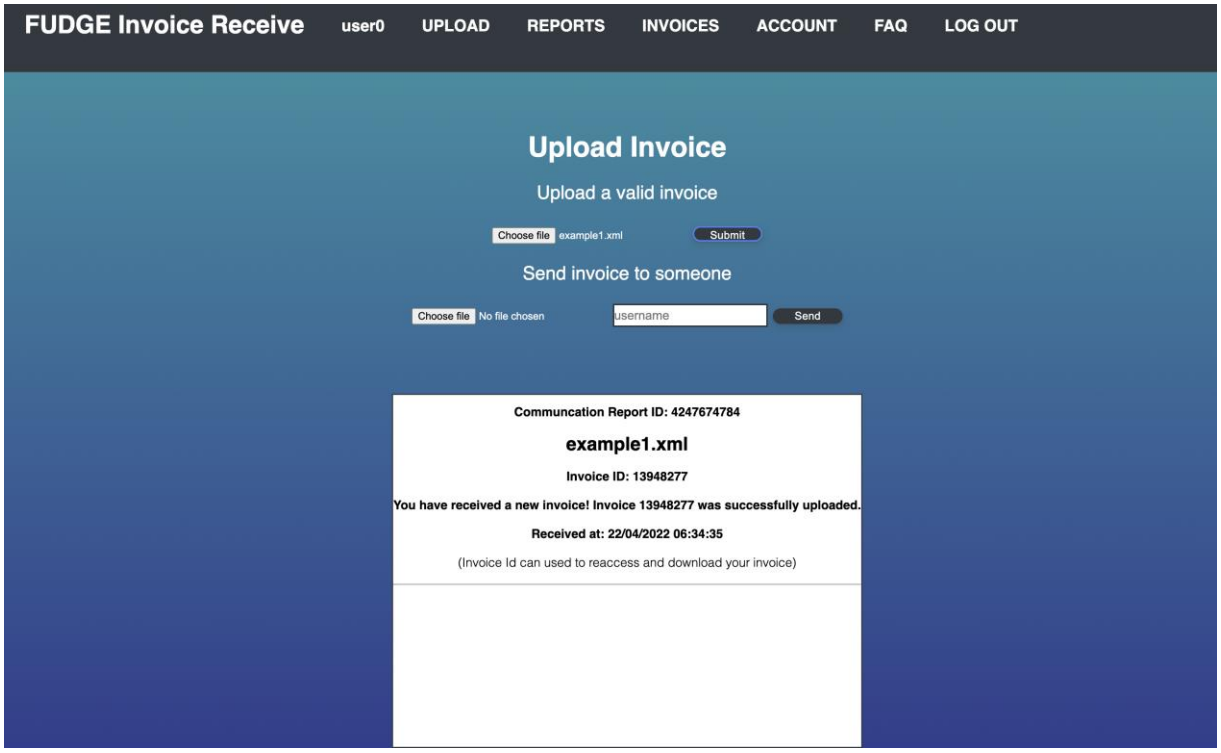


Figure 6 : user is shown details about the xml they upload to the system.

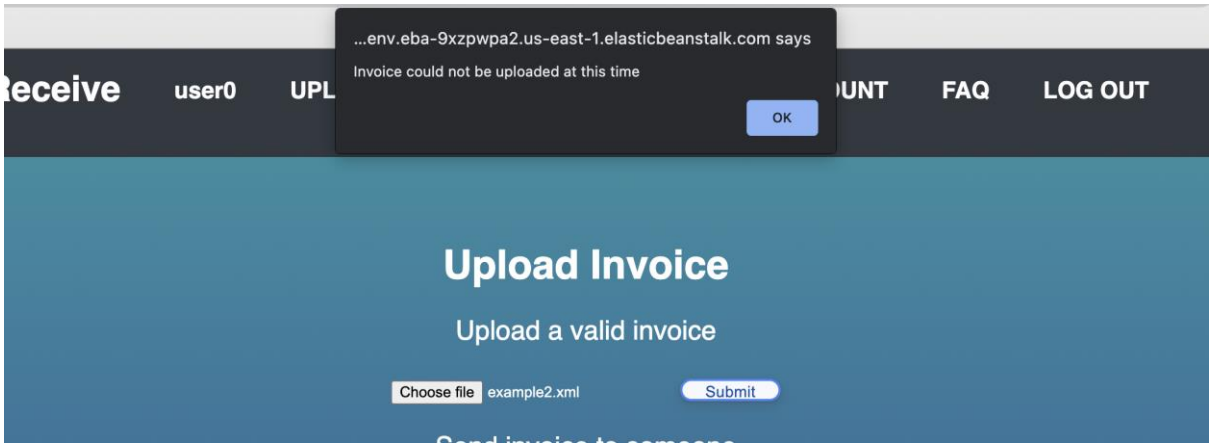


Figure 7: Pop-up when an incorrectly formatted xml file is uploaded.

Such simplistic UI was used throughout the product to provide an accessible and user-friendly application.

3.1.3 Services Layer

Our services layer incorporated 2 APIs. These include the API we had created which was invoice receiving and another team's invoice rendering (https://www.invoicerendering.com/docs).

Our invoice receiving API would receive an invoice and upload it into the system. Consequently, the other team's invoice rendering API would translate the invoice into a readable format for the user to understand and send it over to the transport layer.

By using these 2 APIs, the service layer was able to communicate to the user interface and present the user with the final product. By effectively using each API and extracting the necessary functional requirements from them, the services layer uses a server to communicate to the interface layer, and finally to the client.

3.2 Data Model

As the user interacts with the interface layer, different JSON requests such as POST, GET, PUT and DELETE are sent to the API. The API then calls on a route to perform a request and sends a HTTP response to the front end. Our application's complete data movement throughout the frontend, API, Backend and Data Base can be visualised in figure 8.

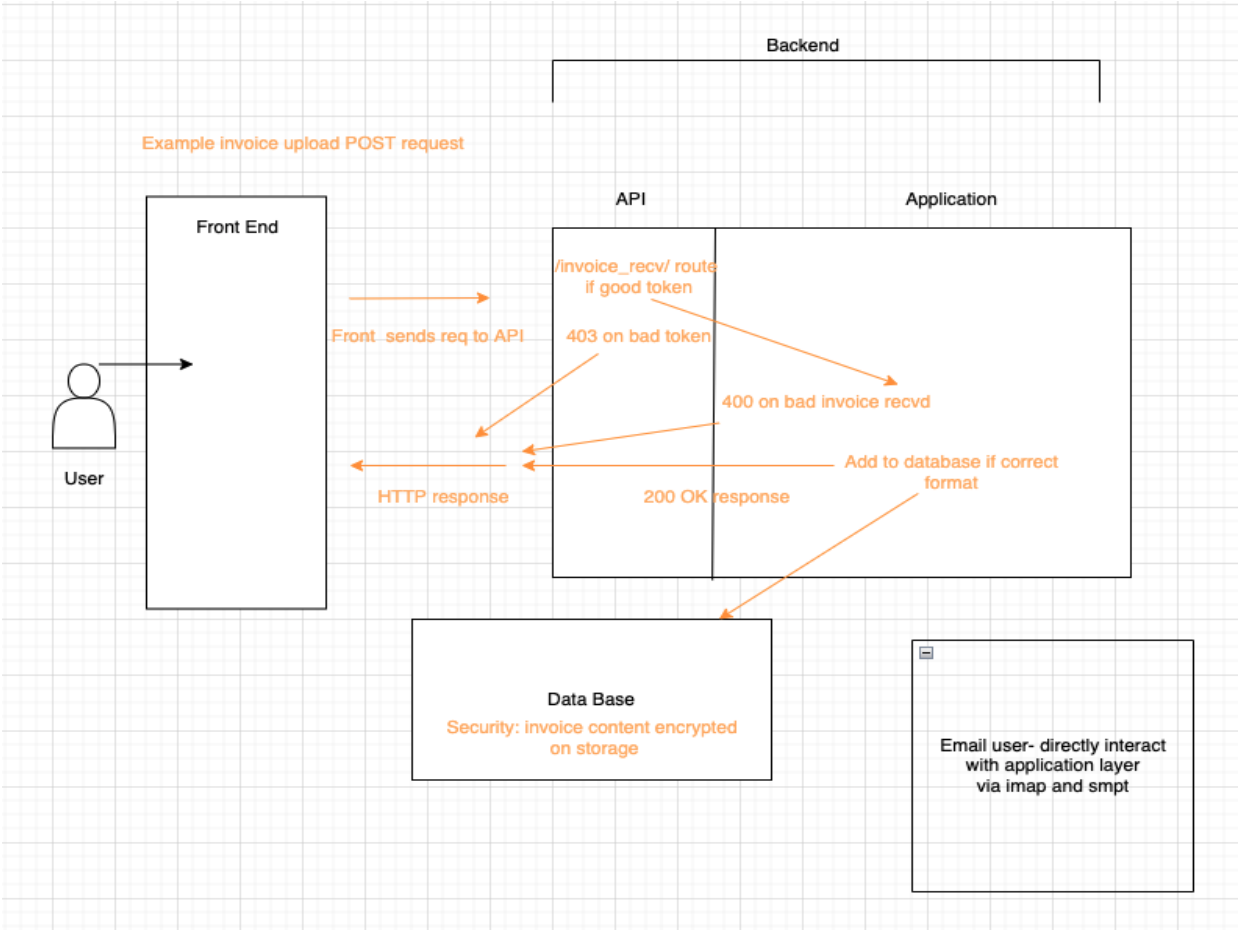


Figure 8: Data model of our application

4.0 API Design

4.1 API Interface

The final interface of the API consists of sixteen routes which the team carefully designed to provide a robust solution to the specified requirements. The routes the team developed focus specifically on receiving, sending, security and user-centric requirements.

Figure 4.1.1 below – Swagger rendering of interface routes. See appendix for full interface specification.

4.2 API Life Cycle

Sprint 1/2:

In sprint 1 and 2 the team's original design consisted of an /invoice_receive/ route, which was a simple file upload. This route was first implemented to cover the basic requirement of invoice receiving.

In addition to the above route, the team also added the option to search communication reports and invoices via invoice name and invoice id. Invoice name searching was introduced as a user is more likely to remember a name than an id. File names are however not unique; hence an invoice/communication report can be uniquely identified via an id. This id can also be used to render and/or delete the invoice via the /invoice_delete/ and /invoice_render/ routes respectively.

Early in the design process the team began to target security requirements. The original solution to this was a token protected access system. This consisted of the user copy pasting a token to access an invoice, much like a password. However, this did not sufficiently satisfy authentication requirements nor was it as user friendly as a human chosen password.

Sprint 3:

To address security issues in sprint two the team implemented a proper authorization and authentication system consisting of password protected user accounts with session token access. From sprint three onwards the routes included /auth_register/, /auth_login/ and /auth_logout/ . The API was then updated to handle this new functionality with invoices and communication reports now linked to the uploader's account. During sprint three a rendering route was also added whereby a raw xml invoice string is converted to a pdf invoice output via a call to an external rendering API (ref 4.2.1 https://www.invoicerendering.com/docs).

Sprint 4:

Now that basic receiving and security requirements had been properly addressed the team began to expand the API's functionality. From sprint 4 onwards more ways to receive and customise accounts were implemented. This included email receiving whereby an invoice/s and its communication report/s would be linked to an existing account included as an addressee or cc'd addressee in an email to invoicerecvapi@gmail.com. If an account for an included email did not exist, the invoice and its communication report would be persisted in an unregistered accounts dictionary for potential future access should the email be registered in the future. This functionality will be updated to only persist unregistered emails and associated data for up to seven days. This will ensure that only users who chose to register with the API have their data stored for longer periods.

The /invoice_send/ route was also added to introduce receiving functionality between registered accounts. Users could now receive invoices via an upload which specified their unique username. The team decided to implement usernames for this functionality as it is more human focused than requiring a randomly generated user id. Usernames are also more secure than requiring a user to disclose their registered email address should they

wish not to. Hence it was crucial to provide at least one way to send to another user that did not require an email address.

The teams also added account customisation routes, focused on practicality and usability. Users can update their username, password and email as well as being able to delete their entire account. Password updating is essential to usability of the product as it allows users to retrieve their account if they forget their password. The ability to update a username and email address further enhances the human focused nature of our API. Initially a user is assigned a random username, hence the ability to customise this is more human accessible. The option to change a user's email address was also deemed necessary should the user begin the process of migrating their business activities to a new email address.

During sprint 4 the API was deployed at http://invoicerecv.us-east-1.elasticbeanstalk.com/ for third party access.

5.0 Final Product

5.1 User Interface Design

Our final product 'Fudge Invoices Inc.' is an e-invoice receiving application in which users are able to use our platform to send their invoices to other users within our platform. Not only does our application allow users to receive e-invoices, it also provides many additional features that enhance our consumers overall user experience when interacting with the website. For example, our application automatically renders UBL-formatted e-invoices to a human readable format that users on the receiving end are then able to download as a pdf. Moreover, our application also conducts a validation check every time an invoice is uploaded. This saves our users time from having to review every invoice manually to ensure it's in accordance with Australian UBL invoice standards. Instead, our application simply displays a communication report that indicates the invoice uploaded was not in the correct format and hence not added to the system.

Here is a publicly accessible link to our web application - http://invoiceplatform-env.eba-9xzpwpa2.us-east-1.elasticbeanstalk.com/

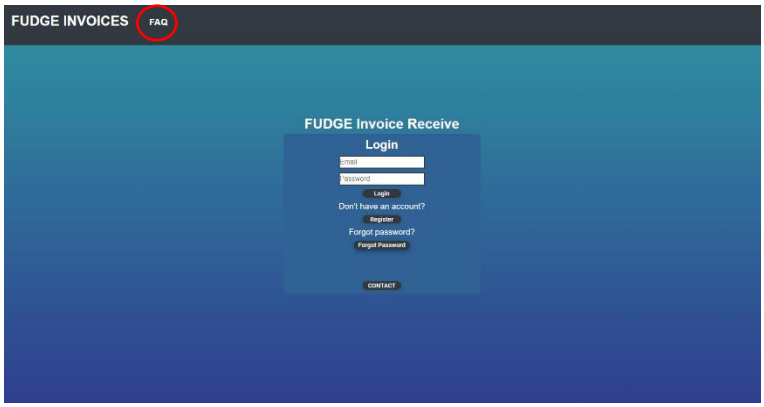
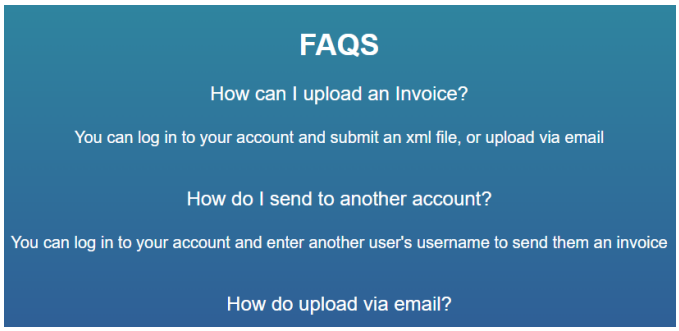


Figure 9: Login page

Once a user enters that link they will be welcomed with a login page (Figure 10).

Figure 10.2: FAQ page

A useful feature to note is the FAQ tab that is always available at the top of every page (circled in Figure 10). Even though we have maintained a clear and easy to use design throughout the website, we have still provided an FAQ page for users to refer to when usure about how to use certain features. Figure 10.2 displays a small snippet of the top of the FAQ page, and Figure 11 is a snippet of our company email we have provided at the bottom of the page for users that wishes to contact us regarding our application.



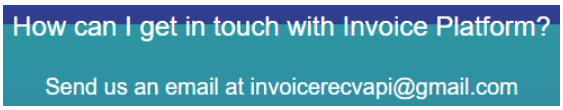
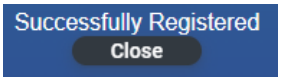


Figure 10: Contact Info at bottom of web page

Figure 11: Registration page

On the login page users may click on the register button to register a new account if don't already have one. The UI design of the registration page can be seen in Figure 12.

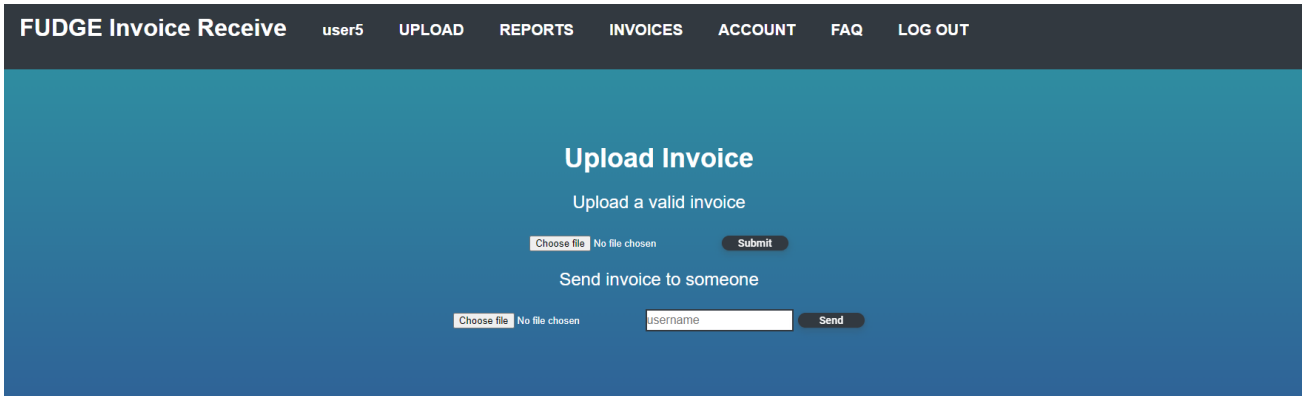
Figure 12: Success Notification



Once registered they will receive a 'Successfully Registered' notification (Figure 13) and be led back to the login page.

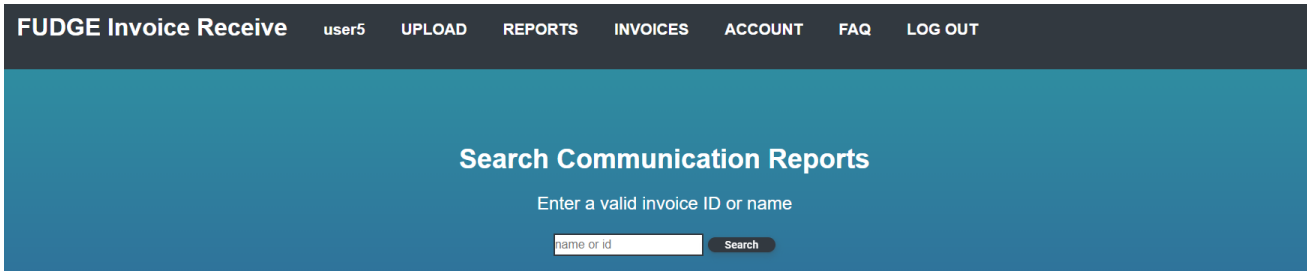
Once logged in, the user will be led to the 'Upload Invoice' page, whose UI design can be seen below. On this page a user can simply upload a UBL formatted valid invoice to received a human-readable formatted invoice for their own use. Alternatively, they could send another user within the platform an invoice using just receiver's username (Figure 14).

Figure 13: Upload page



The ‘Reports’ tab leads to a page that can allows users to search through their own communication reports using a valid invoice_id or invoice_name. (Figure 15)

Figure 14: Reports page



Below are example communication reports when the same invoice is sent to a different user (Figure 16) and uploaded to self (Figure 16.2).

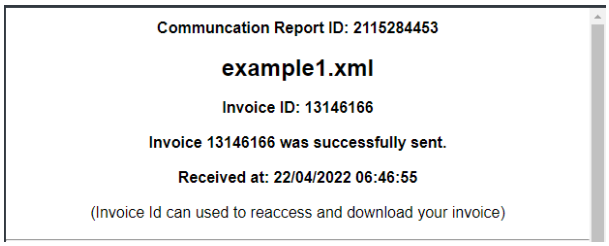


Figure 15: Invoice sent

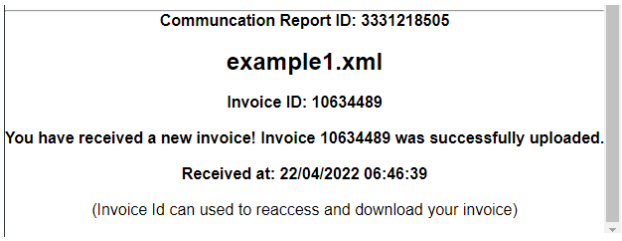


Figure 16.2: Invoice receive

The ‘Invoices’ tab allows users to search through invoices they have received and sent using its invoice_id. A human readable pdf version of the invoice will then appear, which user are able to download or print if they wish. (Figure 15)

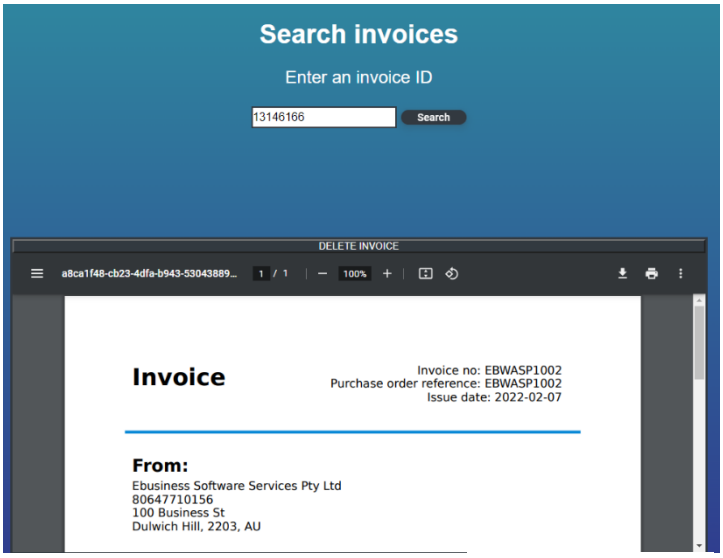
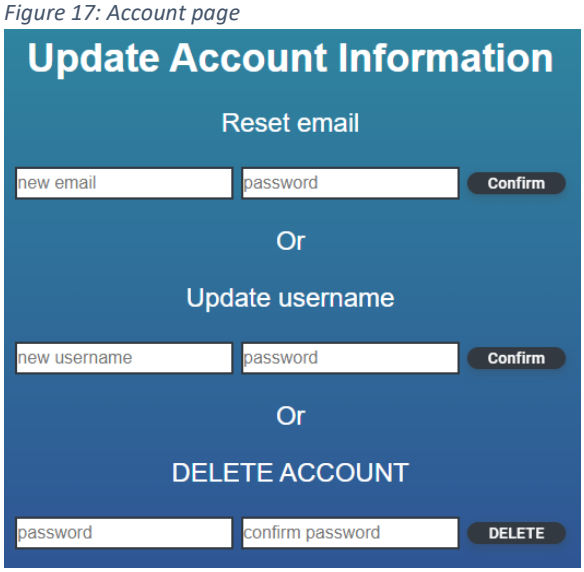


Figure 16: Invoices page

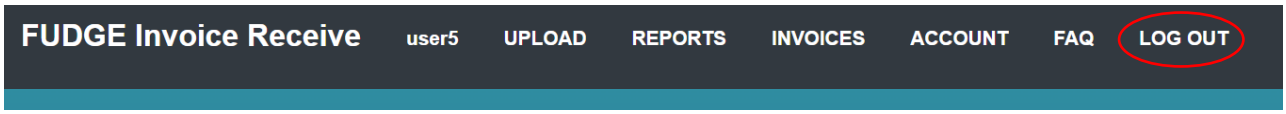
Figure 17: Account page



The ‘Account’ tab takes customers to a page where they may update their account information. For example resetting one’s email, updating a username or deleting their whole account all together. (Figure 16)

Finally, the 'Log Out' tab at the top (Figure 17) allows users to logout of their account and back to the login page (Figure 8).

Figure 18: Log Out tab



6.0 Conclusion

Outlined by Fudge Invoices Inc's finalised product, we can conclude its high efficacy for users to utilise in their invoice interactions. The analysis of this report has implemented pivotal elements such as meticulous product design and extensive research, from APIs to software architecture, to further support the effectiveness of the product's value. The next horizon for this product will involve access to Amazon SMTP services to enable clients to email invoices directly into the system for ease of use. Forward looking, this product has potential to be scaled for use within small to medium enterprises, enabling compliance with new regulation for mandatory e-invoicing, while streamlining invoicing operations.

7.0 Appendix

7.1 Interface

Routes	Description	Method	Inputs	Responses
/api/invoice_receive/	Uploads one or multiple invoices in UBL format and persists the invoice. Returns a communication report with the result of the upload.	POST	Input: • authorization - string, header • invoice_name - string, body • invoice_content - string, body	200: • OK • Output - communication report for uploaded invoice/s ref 1. 400 • Input error in file format • Output - communication report indicating invoice not uploaded. ref 2 403 • access error • bad authorization token

				• Output - communication report indicating invoice not uploaded. ref 2
/invoice_render/	Invoice xml data render to pdf format	POST	Input: authorization, in header xml: raw xml invoice string	200 • successfully rendered pdf ouput 400 • incorrect xml format 422 • invoice could not be validated 403 • access error • bad authorization token
/invoice_send/	Allows user to send to another account on the platform	POST	Input authorization, in header username - string username of intended recipient invoice_content - string content of xml file invoice_name - name of invoice file	403 • Access Error • bad authorization token 400 • invalid username provided • invalid xml file provided 200 • invoice succesfully sent • same output as 200 for invoice receive.
/auth_register/	Allows a user to register for an account to upload invoices	POST	Input: • email, string in body • password, string, in body o len(email) >= 8	200 • {} 400 • Input Error • email not of standard email format or password

				not > 8 chars
/auth_login/	Log in to an existing account to upload invoices.	POST	Input • email, string, in body • password, string, in body	200 • Output - ○ {'token': bytes, 'u_id': integer} 400 • InputError • Incorrect password or email given
/auth_logout/	Log a user out from a session on their account	POST	Input • authorization, in header	200 • Successfully logged out 403 • Access Error • bad authorization token
/invoice_delete{invoice_id}	Delete the invoice with the given invoice_id if it is in the data base.	DELETE	Input • invoice_id, in path • authorization, in header	200 • invoice found and successfully deleted • output: communication report with comm_msg = 'invoice successfully deleted' 400 • InputError • invoice with invoice_id not found in data base 403 • bad authorization token • or invoice with this id is not accessible from this account
/api/search_id/{invoice_id}	Given a valid token, searches	GET	Input:	200 • OK

	for provided invoice_id. If invoice_id is valid, returns communication report corresponding to the invoice Id.		• invoice_id - integer. in path • authorization, in header	• Output - successful communication report corresponding to invoiceId. 400 • Input error, invalid invoiceId • Output - string indicating invalid invoiceId provided 403 • Access forbidden, invalid authorization token • Output - string indicating user not authorised to access resource.
/api/search_name/{invoice_name}	Given a valid token, searches for provided invoice_name. If invoice_id is valid, returns communication report corresponding to the invoice Id.	GET	Input: • invoice_name - in path • authorization - string, in header	200 • OK • Output - list of all communication reports corresponding to invoice_name. 400 • Input error, invalid invoiceId • Output - string indicating invalid invoiceId provided 403

				• Access forbidden, invalid authorization token • Output - string indicating user not authorised to access resource.
/api/invoice_content/{invoice_id}	Given a valid token, searches for provided invoice_id. If invoice_id is valid, returns invoice content corresponding to the invoice Id.	GET	Input: • invoice_id - integer, in path • authorization - string, in header	200 • OK • Output - invoice content corresponding to invoiceId. 400 • Input error, invalid invoiceId • Output - string indicating invalid invoiceId provided 403 • Access forbidden, invalid authorization token • Output - string indicating user not authorised to access resource.
/api/health_check/	Indicates whether API is currently active	GET		200 • API is currently active and working 404 • URL not found- API is not active

				Other errors - api active but not working as expected
/api/auth_reset_password_req/	User can send request to reset password A reset code will be generated and sent to their email address, if an account is found.	POST	Input: email - string, users email for account to reset password	200 • account found and reset code sent to email 400 • input error • No account found associated with this email.
/api/auth_reset_password/	User can enter a reset code and choose a new password for their account.	POST	Input: reset_code - int code generated and sent to user email in above route. new_password - string, new password chosen by user.	400 • Input error • password is less than 8 characters 200 • otherwise
/api/reset_email/	User can enter a new email for the account	POST	Input: authorization - string in header email - string, new email to update for account	200 • email of valid format and not already in use 400 • input error • email of standard email format • email already in use by another account 403 • access error • invalid access authorization
/api/update_username/	User can enter a new user name for their account	POST	Input:	200

			username - string, username to update authorization - string in header	• username successfully updated 400 • Input Error • Output: string indicating username already in use by another user. 403 • access error • invalid access authorization
/api/username/	retrieve username for given account	GET	Input authorization - string in header	200 • OK • Output: • Dictionary containing username for account {'username': username} 403 • Access Error • user could not be authorized, invalid authorization token
/api/delete_account/	Allows user to delete their account	DELETE	Input: authorization - string in header password - string in header	400 • input error • incorrect password for the account 403 • Access Error • user could not be authorized, invalid authorization token 200

				• User account successfully deleted • Ouput {}

ref 1: Below: Successful communication report

```
[{  "comm_time": "08/03/2022 10:50 pm.",
      "recvd": true,
      "invoice_id": 123456,
      "comm_msg": "invoice successfully received",
      "invoice_name": "Invoice from company"
 }]
```

ref 2: Below: unsuccessful communication report. Note no invoice id is assigned to an unsuccessful upload. Note error messages field returns as a string with '<p>' at the start and <\p'> at the end. This will be returned as a string due to how http error messages are handled.

```
[{    "comm_time": "08/03/2022 10:50 pm.",
        "recvd": false,
        "invoice_id": 123456,
        "comm_msg": "invoice was incorrect file format",
        "invoice_name": "invoice.file"
 }]
```

7.2 Links

Products:

API: http://invoicerecv.us-east-1.elasticbeanstalk.com/health_check
API Documentation: https://app.swaggerhub.com/apis/chanteCon/FUDGE_W14A/2.0#/
Application website: http://invoiceplatform-env.eba-9xzpwpa2.us-east-1.elasticbeanstalk.com/

Repositories:

Main github: https://github.com/AishaniM/SENG2021_FUDGE
Deployment github: https://github.com/chanteCon/SENG2021_FUDGE

8.0 Marking Criteria

Criteria	Description
Requirements & Constraint Analysis (20%)	• Do the requirements listed form a complete and accurate specification of the product? • Have the requirements been listed in a consistent and readable format? • Have the requirements been organised in a logical hierarchy? • Have functional and non-functional requirements been differentiated? • Do the requirements outline a specification rather than an implementation? • Is there a definition of what done looks like? • Are the constraints identified logical, valid and justified? • Are the suggested steps taken to exploit constraints feasible?

Software Architecture Design (20%)	• Have all relevant layers of the service been chosen? • Is the application stack logical, well designed and justified? • Is there evidence of deliberation, consideration of multiple options and elimination to decide upon the best ones? • Have factors and constraints been taken into account when deciding upon the stack? • Have the non-functional requirements been accommodated and designed for? • Has a data model been produced which is accurate and demonstrates thoughtful planning? • Have the different layers been written and abstracted appropriately in the code? • Do the sequence diagrams provide a complete and logical overview of the intended use cases? • Do the sequence diagrams incorporate all elements of the architecture and clearly highlight the flow of information? • Are the sequence diagrams well formatted? • Has the API been designed for backwards compatibility? • Is the frontend well-structured and designed?
API Design (20%)	• Does the API design provide a near-complete solution to the specified requirements? • Does the API provide ability to create, read, list, update and delete data in the service? • Have all required fields been included for each endpoint? • Are the endpoint descriptions succinct and understandable? • The interface well formatted and readable? • Proof of plan of API design (Clearly shows how it evolved over time)
Deployment (5%)	• The service has been deployed onto a reliable platform. • The service is available and function for anyone to use on the internet.
Report Structure (15%)	• Strong cover page including key details relevant to the project admin and who is in your group • Well designed table of contents including: List of headings & subheadings An index with page numbers Clickable links • Well formatted body text including headings and subheadings, numbering is best to align with your contents page • Use a nice clean and readable font, size 10 or 12 is good. • Indent paragraphs to show breaks in thought/concept • Use bullet points to break up lists of ideas or items. Avoid indenting bullet points too far into the page. • Inclusion of visual elements such as diagrams, images, graphs. Position them strategically on the page. Caption Figures beneath the figure and tables above the table

	• An overall well written report. Limited spelling and grammatical errors. Maintaining a formal tone. All writing is clear and concise, no long rambles. • An overall well structured report, with introduction, logical sections and a conclusion.
Final Product Design (20%)	• Does the final product design have convincing business value? • Is the final product valuable and relevant to the target audience? • Is the final product competitive with e-invoicing peers? • Is the user interface well designed, human friendly and accessible? • Does the user interface support the concept as a business/product convincingly? • Does the final product, as a solution, sufficiently address the identified requirements? • Is the final product efficient? • From a technical standpoint, is the final product well designed? • Is there evidence of improvement between sprints to the final product?