Control Your Wealth: Cloud Computing Spring2024 Final Project

William Ye
wy2399@nyu.edu
New York University

Ziming Song
zs2815@nyu.edu
New York University

Qinghong Duan
qd2107@nyu.edu
New York University

Wei Li
wl3178@nyu.edu
New York University

***Abstract*—This project report outlines the development of "Control Your Wealth," a comprehensive cloud-based application designed to streamline and enhance personal financial management. Leveraging the power of Amazon Web Services (AWS), our application offers a robust solution for automating the expense tracking and financial reporting processes. Key components of our system include AWS Cognito for secure user authentication, Lambda for serverless computing, DynamoDB for scalable database management, and SQS for effective message queuing. The application efficiently handles receipt scanning, expense record management, and provides visual insights into spending patterns through dynamic reports. With its server-less architecture, the system scales seamlessly to manage varying workloads, ensuring cost-efficiency and robust performance. Our report discusses the architecture, implementation, and benefits of each component, demonstrating the application's capability to transform personal finance management through automation and real-time data processing.**

***Index Terms*—Cloud Computing, AWS, Expense Management, Serverless Architecture, Financial Tracking**

1. Introduction

In today's digital age, managing personal finances remains a complex task for many, primarily due to the cumbersome process associated with traditional expense tracking systems. The rise of e-receipts and digital transactions offers an opportunity to revolutionize this process, making it more seamless and user-friendly. The "ControlYourWealth" project leverages advanced cloud-based technologies to address these challenges by providing a comprehensive solution for expense management.

Our application simplifies the recording of transactions and provides insightful analysis of spending patterns using key AWS components. AWS Textract is utilized for automatic data extraction, interpreting and categorizing content from e-receipts. AWS SES (Simple Email Service) facilitates the management of incoming e-receipts via email, ensuring they are correctly processed and categorized. This integration reduces manual data entry efforts and enhances the accuracy of financial records, while also ensuring data privacy and security.

Key features of the application include:

1. Automated Data Extraction: Utilizing AWS Textract to automatically interpret and categorize information from e-receipts.

2. Dynamic Database Management: Leveraging AWS services to manage database operations efficiently, ensuring data integrity and accessibility.

3. Automated Report Generation: Generating comprehensive financial reports automatically, helping users understand their spending patterns.

4. Efficient Email Processing: Using AWS SES for handling incoming e-receipts, streamlining the data entry process and ensuring timely updates to financial records.

In the following sections of this report, we will delve into the architecture design of our application, detailing how each component interacts within the AWS ecosystem to facilitate seamless operation. We will also discuss the implementation strategies that were employed to realize the features of the application, followed by testing methodologies that ensure the robustness and reliability of the system. The source code can be found in [1]

2. Architecture Design

This project adopts a server-less and event-driven architecture, utilizing various AWS services to create scalable and cost-effective application. By maximizing the use of AWS scalable services such as Lambda, DynamoDB, API Gateway as well as asynchronous event handling, the application is designed to handle variable work loads while keeping the cost low. In this section, we will describe in detail the architecture of the system, the functionalities of each component, as well as an overall dataflow of the system.

2.1. System Components and Functionality

As shown in Figure 1, the architecture integrates various AWS services to build a scalable systems that manages to handle user authentication, data processing, interaction and communications between server-less components. The description of the components and functionalities is as follow:

1) **Authentication Service by Cognito**: AWS Cognito manages user authentication and authorization. It provides featuers like user login, registration as well as

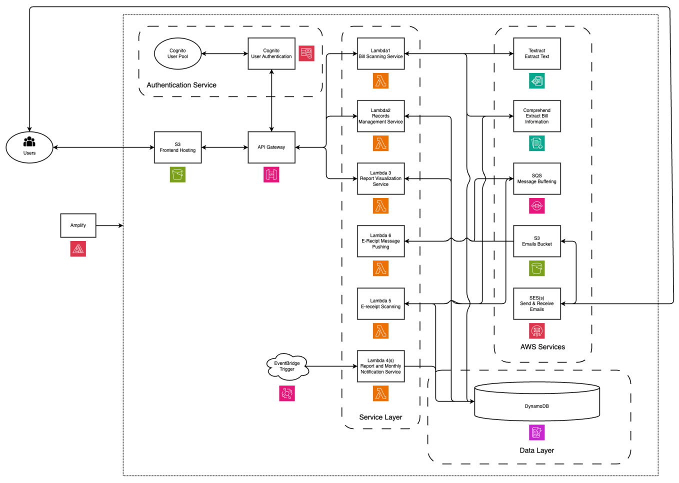


Figure 1. System Architecture

more advanced ones such as Multi Factor Authentication and user data encryption.

2) **Frontend Hosting by S3 and Amplify**: S3 is used as the container for hosting the static web content consisting of HTML, CSS and JS. AWS Amplify simpifies the process of deploying and managing the frontend by automating the deploying process.
3) **API Gateway**: It acts as the entry point for all backend services, efficiently routing API calls to appropriate AWS Lambda functions based on the request path and method. API Gateway also handles request throttling and ensures API consistency.
4) **Service Layer by Lambda Functions**: Lambda is a server-less computing service that allows services to be deployed without the need to manage servers or runtime enviornment. The service layer of this system consists of several Lambdas, each performing specific tasks:
   a) **Lambda 1, Bill Scanning Service**: Utilizes TextTract and Comprehend to scan the bill images uploaded by users. It then record the expense into the database.
   b) **Lambda 2, Records Management Service**: Manages existing expense records and perform update or deletion onto the database.
   c) **Lambda 3, Report Visualization Service**: Periodically sends email to users concluding their monthly expenses. It also reminds users to login and check the charts and table available.
   d) **Lambda 4, Report and Monthly Notification Service**: Periodically sends email to users concluding their monthly expenses. It also reminds users to login and check the charts and table available.
   e) **Lambda 5, E-receipt Scanning Service**: Processes email-format e-receipts forwarded by users and incorporates with AWS Textract to extract total amount of the record.
   f) **Lambda 6, E-receipt Message Pushing**: Pushes messages to AWS SQS indicating that there are newly arrived emails to process.
5) **SQS for scalability**: SQS handle a high volume of messages without losing messages or requiring other services to be always online. SQS scales automatically with the application's demand, making it a cost-effective solution for managing peak loads efficiently.
6) **EventBridge for Event-Driven Integration**: Is triggers an event monthly that invoke the Lambda4 for sending notifications to users without manual intervention.
7) **DynamoBD for Data Store**: AWS DynamoDB acts as the database in the system, storing the information of users' expense records. The data schema is described as follow:
   a) **Username**: Partition key. This string type of attribute is the unique identifier of each user. It is also stored inside Cognito's user pool.
   b) **Timestamp**: Sort Key. This attribute is used to order the record data of each user according to time, allowing a faster fetching time.
   c) **Amount**: a floating number, recording the total amount of expense of this record.
   d) **Category**: a string value, recording which kind of expense one record belongs to.

2.2. Data Flow

The data flow begins with the user authentication by AWS Cognito. After users are authenticated, they are redirected to the dashboard at the frontend hosted by S3. Requests are then sent from frontend through API Gateway, which ensures that only authenticated requests can access the service layer. The lambda functions in the service layer then process the requests, calling other AWS services' APIs as needed, and finally perform actions based on the request such as sending back response or modifying the database. One exception would be when users are forwarding e-receipts to the system, the data pass through API Gateway and go directly to SES, which pushes messages to SQS for later polling and processing by Lambda5. Lambda5 then perform actions of extracting data and writing database. The detail of each service logic will be presented in section 3.

3. Implementation

3.1. Receipt Image Upload and Processing Service

Implementing the service for uploading and processing receipt images involves several critical AWS services to manage the data flow efficiently. This section outlines the step-by-step process of configuring and implementing this service:

3.1.1. Image Upload via API Gateway and Lambda.

1) **Lambda Function Integration.** We configure an API Gateway to receive POST requests containing images encoded in base64 format. This gateway acts as the entry point for the frontend application to upload receipt images.
2) **API Gateway Configuration.** The API Gateway routes the incoming requests to a Lambda function. This function is responsible for decoding the base64-encoded images, performing preliminary validations, and storing them in an S3 bucket for further processing.

3.1.2. Processing with AWS Textract.

1) **Triggering Textract from Lambda.** Once the image is stored in S3, the same Lambda function triggers AWS Textract. Textract is utilized to extract text from the receipt images, focusing on identifying monetary amounts related to purchase totals.
2) **Optimizing Text Extraction.** We implement specific logic within the Lambda function to interpret the Textract response. The function searches for keywords related to total amounts, such as ”total,” ”grand total,” etc., and selects the highest value when multiple candidates are detected to ensure accuracy.

3.1.3. Data Management with DynamoDB.

1) **Storing Extracted Information.** After processing the receipt image, the Lambda function stores the results in a DynamoDB table. Each entry includes the username (retrieved from session storage), a timestamp in New York timezone (converted from UTC), the extracted total amount, and the category selected by the user.
2) **Timestamp and User Information.** The username and timestamp are crucial for tracking the submission context of each receipt, allowing for personalized data retrieval and time-based querying.

3.1.4. Frontend Integration and User Interaction.

1) **Secure Image Upload.** The frontend application uses Dropzone.js configured to handle image files and encode them in base64 before sending. This setup ensures that user interactions are smooth and secure.
2) **User Feedback and Error Handling.** The frontend provides real-time feedback on the upload process, successful data extraction, and errors. This immediate feedback enhances user experience by ensuring users are informed of the status of their uploads.

3.1.5. Security and Access Management.

1) **Securing API Gateway.** We use AWS IAM roles and policies to secure the API Gateway. Access tokens and identity verification are managed via AWS Cognito, ensuring that only authenticated users can upload receipt images.
2) **CORS Configuration.** To allow requests from the frontend hosted on different domains, CORS is configured on the API Gateway to permit such cross-origin requests, addressing common web application security concerns.

3.2. Records Management Service

We implement the records management service to make it easier for users to change their expenses. Instead of integrating all crud operations into one lambda function, we design four respective lambda functions for each operation.

3.2.1. Create. For the Create lambda function, we collect the category and corresponding amount in the front-end by user’s input. Then we create a timestamp on the server when the user clicks the add button and add the three fields to the post request body to API Gateway. For invalid input, we also set filter part to eliminate this situation.

3.2.2. Read. We set up a GET request in API Gateway to achieve basic Read operation. When Read lambda function is invoked, it uses username for query and returns the corresponding user data.

3.2.3. Update. The implementation of Update operation is similar to Create. We achieve Post method in API Gateway. In addition to category and amount, we also need the original timestamp for query to locate the original data. Then we can do the simple update operation in lambda function.

3.2.4. Delete. We implement delete method in API Gateway, we store the timestamp in query parameters. When the delete lambda function is invoked, it deletes the data in DynamoDB by username and timestamp.

3.3. Report Visualization Service

In order to give users a better experience for management and understand their monthly bill, we design charts to visualize proportions of different categories. With visual reports, users can quickly pinpoint areas of overspending or identify potential cost-saving opportunities.

To visualize report, we set up the following: (1)Setting up based on web front-end, implemented pie charts using JavaScript, HTML, CSS, jQuery Sparklines [2]. (2)Setting up API Gateway to implement GET request ’/report/month’, which is used for getting bills of a specific month. (3) Setting up Lambda 3 which trigger by the API Gateway to get monthly bill from DynamoDB.

In this section, we will introduce these process in detail:

3.3.1. Front-end Design. The visualization pie charts mainly shown on two web pages. One is *’index.html’*, the other is *’analytics.html’*. In *’index.html’*, we display current month bill report. In *’analytics.html’*, users can choose the report display of a certain year and month. But the basic logic and data flow behind are same.

1) **Get parameter of month and year.** For the two pages mentioned before, the only difference is the parameter of month and year. In *’index.html’*, year and month is given by system, system will add current time and pass it to API Gateway. In *’analytics.html’*, user selected month and year will be passed to API Gateway.

2) **GET request to API Gateway, and parse response data.** In API Gateway, setting up a GET request in *'/report'* path which name is *'month'*, the request *'URL query string parameters'* are *'month'* and *'year'*. The request integrated with Lambda 3. CORS is configured on the API Gateway to enable cross-origin requests.
3) **Setting up Lambda 3.** This Lambda function is triggered when receive a GET *'/report/month'* request. Lambda 3 will get data from DynamoDB according to URL query string parameters. The response data provide that month each category's amount and generate a report.
4) **Visualize data in pie charts** After getting response data, front-end parse the response data and get each catagory's amount, then using Sparklines [2] to visualize each category's proportion pie chart of a specific month.

3.4. Report and Monthly Notification Service

We also design and implement a feature that users can receive their last month report on the first day of a new month. In order to set up monthly report notification service, we set up the following components:

3.4.1. Key Components.

1) **Lambda 4, Report and Monthly Notification Service.** This Lambda function is triggered at the first day of each month in UTC timezone. First, Lambda 4 get all users in the *Cognito User Pool*. Then for each user, the Lambda 4 get each user's last month bill report from DynamoDB and generate an month report then send it. The logic of Lambda 4 is shown in Algorithm 1.
2) **Cognito.** Use *cognito_client.list_users* to get all verified users and their email.
3) **SES.** SES is used to sent monthly bill report. And only verified email identities can receive monthly bill notification report.
4) **EventBridge Trigger.** EventBridge trigger is connected with Lambda 4. It triggered at the first day of each month. And different user receive the email at random time in that day, which can reduce SES load.

Algorithm 1: Lambda 4

```
  Data: event, context
  Result: Response from Lambda function
1 $users \leftarrow$ cognito.UserPool
2 for $each\ user\ in\ users$ do
3     $year, month \leftarrow$ get_year_month()
4     $message \leftarrow$
        get_message_dyno($user["id"], year, month$)
5     send_email($message, email, year, month$)
```

3.5. E-Receipt Scanning Service: Receiving Emails

To implement the service of scanning e-receipts forwarded by the users, we need to set up the following: (1) Setting up the email receiving service on AWS SES (2) Setting up rule set and the receipt rules to transfer emails to AWS S3 bucket (3) Configuring and implementing the data flow of processing the emails. In this section, we will discuss step by step through these process:

3.5.1. Receiving Email From User. We can configure AWS SES to receive email on behalf of our domain. To allow SES to perform such action, we need to satisfy several prerequisite:

1) **Creating a domain identity.** To create a domain identity, we simply need to sign in the SES console, find the Create Identity option in the console, and fill out the name of the domain.
2) **Verifying the ownership of a domain by publishing DNS records.** In this application, we uses a domain hosted on Tencent Cloud, which allows us to manage the records in the DNS table as needed. To enable SES to discover the verifycation information set on the Domain, we configured Easy DKIM on the hosted domain, which requires us to publish a list of CNAME records formatted as Name:Value in our DNS table. After configuring the table, SES will detect the changes and validate the records.
3) **Permitting SES to receive email for your domain by publishing an MX record.** A mail exchange(MX) record is a configuration that specifies which mail servers can accept email that's sent to our domain. To have Amazon SES manage your incoming email, you need to add an MX record to your domain's DNS configuration. The MX record that you create refers to the endpoint that receives email for the AWS Region where you use Amazon SES. For example, the endpoint that we used for US-East-1(N. Virginia) is *feedback-smtp.us-east-1.amazonses.com*.

Now, the AWS SES is reday to receive emails transferred by our hosted domain.

3.5.2. Setting up the email receiving rules. The extend of processing the received email is determined by the custom instructions specified by the rule sets, which are in the form of *Receipt Rules* or *IP Address Filters*. In this application, we chose the former, which allows us to define advanced processing such as delivering mails to S3 bucket, publishing them to AWS SNS topics, etc. Our receipt rule is configured as:

1) Only receiving the emails sent to the only *service email*.
2) Delivering the emails to S3 bucket after receiving them.

Instead of invoking a Lambda function to process the emails, our system is set to deliver emails to an S3 bucket for system scaling and decoupling, which will be explained in the following section.

3.6. E-Receipt Scanning Service: Processing Emails

In this section, we will start by introducing the data flow from when an email is received to when the e-receipt is extracted and a new record is inserted into DynamoDB. Then, we will discuss the implementation of the key components in this data flow.

3.6.1. Data Flow. To turn an email containing an e-receipt into a record in the database, the following events happen sequentially:

1) The user receives an e-receipt in his/her inbox, and then forwards that receipt to a service email address associated to the system's hosted domain.
2) The hosted domain transfers the email to SES, which will deliver the email as a file to the S3 bucket(Emails Bucket).
3) The S3 detects that an object is put, triggering the Lambda6. This Lambda then constructs the put event into a message {bucket_name, key_name} which indicates where the email file lies and which file contains the email and pushes it into SQS.
4) Lambda5 is then triggered by SQS's lambda trigger, polling the messages in a batch manner, calling Textract API for the expenses' amount.
5) After the expenses' amount are extracted, Lambda5 will pick the total amount, extract email senders(users), query Cognito for users' identification and then construct an item to put in DynamoDB.
6) Finally, a new record of user's expense is inserted into the database table.

3.6.2. Key Components. As the configuration of SES was explained in Section 3.5.1, we will focus on the rest if the components in the data flow, which are both of the Lambdas and the SQS.

1) **Lambda 6, E-Receipt Message Pushing.** This Lambda function is triggered when a new object is put into the S3 bucket. The only task of this Lambda is constructing the S3 PutEvent into a message and push it into SQS. Algorithm 3 shows the logic of this Lambda.
2) **Lambda 5, E-Receipt Scanning.** This Lambda function is triggered by SQS events, which pass a batch of message to the Lambda. The lambda will query Cognito for the user identity for further record construction. By calling the AWS Comprehend API, it is able to obtain a list of candidate amount(in practice, these are presented as type "QUANTITY" by Comprehend) for each e-receipt. Then, the total amount is extracted by picking the last amount for simplicity. The logic of Lambda 5 is shown in Algorithm 2.
3) **SQS and Lambda Trigger.** The SQS decouple the process of receiving emails and processing emails, allowing the workflow of the e-receipt scanning service to be asynchronous, thus improving scalability. To improve the performance of Lambda5, we focused reducing the code-start times of it. We configured the lambda trigger to invoke Lambda5 with a batch of emails with a batch window. Via this approach, a batch of emails can share one code start time. The batch window also allows the system to accumulate some emails before passing them to Lambda5, reducing the costs by minimizing the number of Lambda invocations needed.

Algorithm 2: Lambda 5

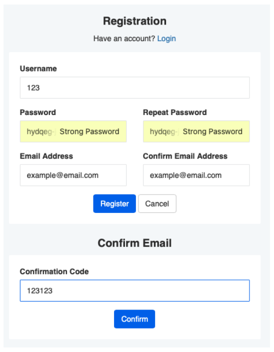
```
Data: event, context
Result: Response from Lambda function
1 $records \leftarrow$ event["$Records$"]
2 for each $record$ in $records$ do
3     $bucket\_name, object\_key \leftarrow$
        deconstruct_message($record$)
4     $message \leftarrow$
        s3_client.get_object($bucket\_name, object\_key$)

5     $user \leftarrow$ cognito.recognize($message$["$From$"])
6     $entities \leftarrow$
        comprehend.detect($message$["$Body$"])
7     $total \leftarrow entities$["$QUANTITY$"][$-1$]
8     $item \leftarrow$ construct_item($user, total$)
9     dynamo.put($table\_name, item$)
```

Algorithm 3: Lambda 6

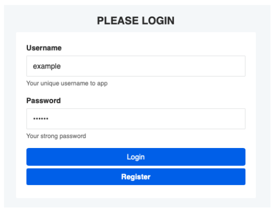
```
Data: event, context
Result: Response from Lambda function
1 $bucket\_name \leftarrow$
    $event$["$Records$"][$key\_to\_bucket\_name$]
2 $object\_key \leftarrow$
    $event$["$Records$"][$key\_to\_object\_key$]
3 $sqs\_message \leftarrow$
    $constrct\_message\{bucket\_name, object\_key\}$
4 $response \leftarrow$
    $sqs\_client.send\_message(queue\_url, sqs\_message)$

5 $lambda\_response \leftarrow$
    $construct\_lambda\_response(response)$
6 return $lambda\_response$
```



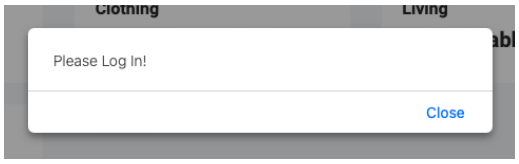
Username: 123
Password: hydqeg- Strong Password
Repeat Password: hydqeg- Strong Password
Email Address: example@email.com
Confirm Email Address: example@email.com
Confirmation Code: 123123

Figure 2. Login and Register: Registration



Username: example
Your unique username to app
Password: ••••••
Your strong password

Figure 3. Login and Register: Login


Clothing Living

Please Log In!

Close

Figure 4. Login and Register: Login Protection

4. User Perspective

In this section, we will demonstrate how the application will work from a user perspective. The front-end template we referenced the template [3]

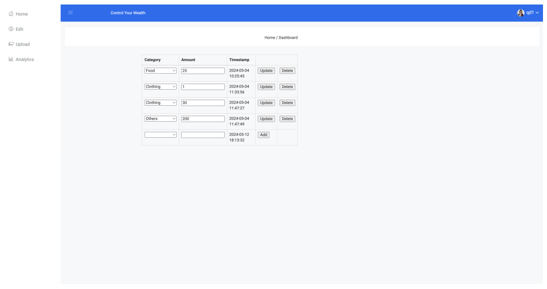
4.1. Login and Register

4.1.1. Register. From the welcome page, users are able to find the register button. Clicking on the button, users will be asked to fill a register form as shown in Figure 2. They will then receive an email verification code. By submitting all these information, users will be successfully registered.

4.1.2. Login. All the pages and back-end URIs except for those about login and register are protected by Cognito, shown in Figure 4, which uses JWT token to perform authentication and authorization. By providing username(the one a user can specify when register) and password, as shown in Figure 3, user can login and will be redirected to the dashboard.

4.2. Records Management

In the Edit section, users change their expenses conveniently as shown in Figure 5.

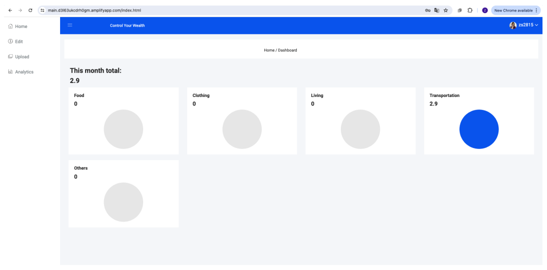


Category	Amount	Timestamp		
Food	25	2024-05-04 10:25:45	Update	Delete
Clothing	1	2024-05-04 11:33:56	Update	Delete
Clothing	30	2024-05-04 11:47:27	Update	Delete
Others	200	2024-05-04 11:47:49	Update	Delete
		2024-05-12 18:13:32	Add	

Figure 5. Records Management

4.3. Monthly Report Visualization

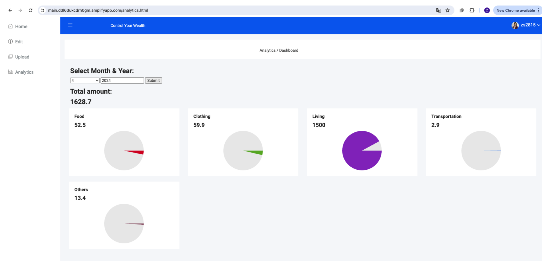
After login, the front-end will route to *'index.html'*. In this page Figure 6, we can see current month bill based on each type category's proportion. The categories contain *'Food'*, *'Clothing'*, *'Living'*, *'Transportation'*, *'Others'*. Similarly, user can click left navigation bar jump to page *'analytics.html'* Figure 7 and select a specific month and year to view that month bill pie charts.



This month total:
2.9

Food 0
Clothing 0
Living 0
Transportation 2.9
Others 0

Figure 6. Monthly Report Visualization: current month



Select Month & Year:
4 2024

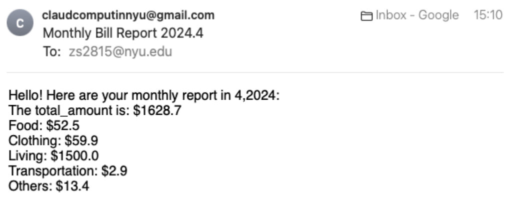
Total amount:
1628.7

Food 52.5
Clothing 59.9
Living 1500
Transportation 2.9
Others 13.4

Figure 7. Monthly Report Visualization: 2024 April

4.4. Report and Monthly Notification

At the first day of each month, users will receive an monthly bill report email looks like Figure 8


claudcomputinnyu@gmail.com
Inbox - Google 15:10
Monthly Bill Report 2024.4
To: zs2815@nyu.edu

Hello! Here are your monthly report in 4,2024:
The total_amount is: $1628.7
Food: $52.5
Clothing: $59.9
Living: $1500.0
Transportation: $2.9
Others: $13.4

Figure 8. Monthly Report Notification Email

4.5. E-Receipt Scanning

To access the E-Receipt Scanning Service, users will simply forward any e-receipt in their inbox of the email address they used to registered for the application. Then, all the processing are automatically done by the application. Without minutes, the record will then appear in the dashboard with a tag "E-RECEIPT". Following is an example of recording a $1.5 fee by E-Receipt Scanning:

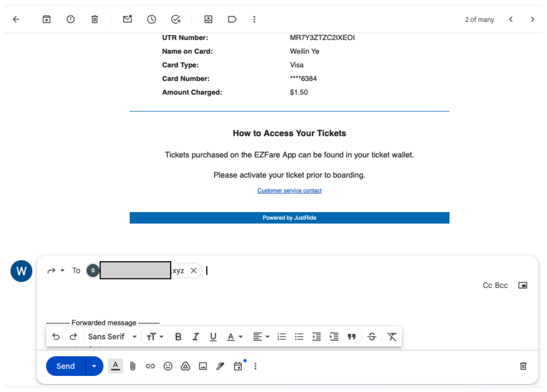
1) Forwarding the email to the service email address: Figure 9
2) View the record in the dashboard: Figure 10

References

[1] Control Your Wealth source code. https://github.com/Iris-Song/control-your-wealth.

[2] jQuery Sparklines. https://omnipotent.net/jquery.sparkline.

[3] Kiaalap Admin Dashboard. https://github.com/puikinsh/kiaalap.



UTR Number:	MR7Y3ZTZC2IXEOI
Name on Card:	Weilin Ye
Card Type:	Visa
Card Number:	****6384
Amount Charged:	$1.50

How to Access Your Tickets

Tickets purchased on the EZFare App can be found in your ticket wallet.

Please activate your ticket prior to boarding.

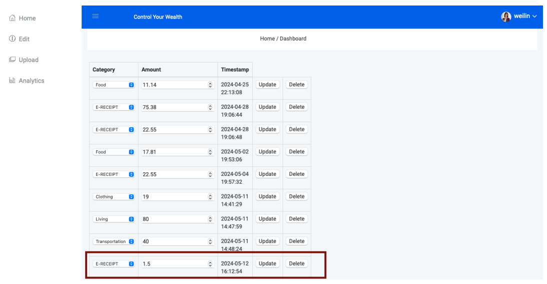
Customer service contact

Powered by JustRide

To [redacted]xyz

---------- Forwarded message ---------

Figure 9. Scanning E-Receipt: Forward Receipt



Category	Amount	Timestamp		
Food	11.14	2024-04-25 22:13:08	Update	Delete
E-RECEIPT	75.38	2024-04-28 19:06:44	Update	Delete
E-RECEIPT	22.55	2024-04-28 19:06:48	Update	Delete
Food	17.81	2024-05-02 19:53:06	Update	Delete
E-RECEIPT	22.55	2024-05-04 19:57:32	Update	Delete
Clothing	19	2024-05-11 14:41:29	Update	Delete
Living	80	2024-05-11 14:47:59	Update	Delete
Transportation	40	2024-05-11 14:48:24	Update	Delete
E-RECEIPT	1.5	2024-05-12 16:12:54	Update	Delete

Figure 10. Scanning E-Receipt: Result