SUMMER PROJECT REPORT

INTEGRATION OF PAYMENT GATEWAY IN CURRENT MESS SYSTEM

UNDER GUIDANCE OF DR. DWAIPAYAN ROY

SUBMITTED BY: ABHAY SAXENA

IISER KOLKATA

Indian Institute of Science Education and Research Kolkata

I am writing to express my sincere gratitude to **Dr. Dwaipayan Roy** and **Nimish Sharma** for their invaluable contributions to my summer project report. Dr. Roy's expertise and suggestions were essential to the completion of this project, and Nimish's contribution was essential to its success. Their feedback has been invaluable. Their contributions have made this project.

~Abhay Saxena

Contents

Section 1

Introduction to Payment Gateway Integration

1.1 Aim Of This Project

The findings and results of a project aimed at learning how to create a frontend web page using frontend web development (HTML, CSS, JS) and backend python-based server framework Django are presented in this Summer Project Report. Additionally, a payment gateway is integrated, enabling wallet top-up capabilities. The project's goals, approaches, difficulties, outcomes, and learned skills are described in this report.

The report's opening emphasises the value of integrating a payment gateway and developing a frontend webpage for the website. It emphasises the significance of these abilities in creating valuable and approachable online apps. Creating a frontend webpage and integrating a payment gateway are described in an outline of the project's goals. The paper then reviews the project's procedures, including the coding and experimental methods, learning strategies, and tools employed.

The project's technologies are comprehensively covered, concentrating on HTML, CSS, JS, and Django. The function and importance of each technology in web development are presented to provide a basic grasp. Step-by-step instructions are provided for the development process, which covers several topics, including frontend design, which includes HTML structure and CSS styling, the construction of interactive features using JavaScript, and integrating a payment gateway for wallet top-up capability.

The paper also discusses the difficulties found during the project and how to solve them. It considers the essential lessons discovered through these encounters. The project's outcomes and accomplishments are assessed, along with the generated frontend webpage's functionality, usability, and responsiveness. The successful wallet top-up feature connection with the payment gateway is emphasised and explored.

It offers recommendations for extra study material, skill-building exercises in frontend web development and payment gateway integration, and additional features or functionalities that could be included.

This Summer Project Report thoroughly reviews the project's goals, approaches, difficulties, outcomes, and learned skills. It highlights the practical implementations of these abilities. It is a valuable resource for learning how to design a frontend webpage and integrate a payment gateway for wallet top-up functionality.

1.2 Significance

The significance of payment gateway integration in the context of the student mess wallet recharge process cannot be overstated. By incorporating a payment channel, students may quickly replace their trash wallets, leading to a more convenient and streamlined experience. This relationship benefits the person in charge of entries in the mess wallet and the pupils.

Students can quickly recharge their accounts thanks in part to the payment gateway link. In the past, students had to manually travel to the mess office or speak with the appropriate person to make cash payments and update their wallet balances. This process is time-consuming and inconvenient, especially when several students must recharge their wallets simultaneously during busy periods. However, thanks to the payment gateway connection, students can use various payment options like credit/debit cards or Internet banking to replenish their trash wallets online. Giving students a hassle-free and straightforward way to top off their balances anytime they want, from anywhere, removes the need for in-person visits and reduces wait times.

The process of filling out entries for the person in charge of managing the balances of numerous wallets is also made more accessible by adding a payment method. The person in control had to manually update wallet balances, monitor cash transactions, and precisely record each student's recharge before this. This manual process required much time and was prone to human error and inconsistency. Once a student makes an online payment, the system might instantaneously update wallet balances if a payment gateway is integrated.

The cost-effectiveness of integrating a payment gateway is enhanced by several factors. First, it does away with the necessity for managing currency transactions, which may be risky and time-consuming.

Switching to online payments significantly decreases the risks associated with handling and managing cash. Setting up and maintaining a payment gateway is typically less expensive than manually handling cash. As the costs of the physical infrastructure, human resources, and cash management are drastically reduced, the institution will see long-term cost reductions.

1.3 Objectives

We first list out the purpose of this project to assign with some objectives.

- To evaluate the payment gateway integration's usability,functionality, and affordability.
- To get beneficial knowledge in integrating payment gateways and web development that can be applied to future projects.
- To design a responsive, functional, and intuitive user interface for a front-end website.
- Automate the process of updating wallet balances, and integrate the payment gateway.
- To master web development in Django, HTML, CSS, and JS.
- To become familiar with wallet features at a large scale and understand the current mess system.
- The user experience is enhanced by giving students access to a quick and secure online wallet recharge mechanism.

Our priority should be creating an aesthetically pleasing, user-friendly frontend webpage with a visually appealing design and straightforward navigation. Hence we should use responsive design concepts to offer a consistent user experience across devices and screen sizes. We can also reduce loading times by boosting the performance of the frontend web page and the speed of payment gateway integration. We Shall also create interactive JavaScript components to increase user involvement and provide dynamic functionality. Including a reputable payment processor that allows a variety of payment methods for wallet top-ups would provide students with a safer and easier experience. Allowing for real-time, faultless synchronisation of wallet balance and payment gateway updates would provide the advantage of automation to students. We should provide dependable error management and validation methods to ensure exact and secure payment transactions. Ascertain that the frontend webpage is compatible with all major browsers and mobile devices.

Section 2

Languages And Frameworks

2.1 Overview Of Tools Used

Before generating, we used Hypertext markup language (HTML) to create specific instructions for a web page's design, type, format, structure, and makeup. It is mainly used to make the essential structure of a page accessible to the user and to assign specific markups to each element for future reference by other tools. HTML is used to assign our input components and names. The same is true for all other information, buttons, etc.

CSS (Cascading Style Sheets) is also used in the project to improve the look of a web page. By including meaningful CSS styles, you may make your page more appealing and enjoyable to read and utilise for the end user.
JavaScript is the programming language used for our portal's web browser rendered section. The Document Object Model API (DOM) allows you to alter any element in any way you wish. When integrating a payment gateway, it is critical to facilitate user interaction and input validation. It allows for the management of payment gateway replies, such as success or failure signals, and the initiation of further actions in response to these responses.
Python, on the other hand, is the programming language used for the backend. The backend is the website's server side. It saves and organises data while ensuring that everything on the client side of the website functions properly. It is a website section you cannot view or interact with. To put it simply, for our servers to launch a website on the World Wide Web, we want a web framework meant to facilitate the construction of online applications such as web services, web resources, and web APIs and is thus typically approved by most browsers. Django is a free and open-source Python-based web framework that we are utilising in our project and was previously used for the mess portal's web platform. Django offers essential features for managing form submissions, processing payment requests, and securely interacting with the API of the payment gateway provider. It follows a model–template–views architectural pattern.

2.1.1 Model-view-template architecture

- A **Model** is an object that defines the data structure of a Django application. It is in charge of keeping the entire application's data and provides numerous techniques for adding, updating, reading, and deleting data from the database.
- A **View** is an HTTP handler that accepts HTTP requests, processes them, and returns the HTTP response. It uses Models to retrieve the data needed to fulfil the request and Templates to render it on the user interface. It can also dynamically generate an HTML page using an HTML template and populate it with data from the model.
- A **Template** is a text file that defines the user interface's structure or layout. The text file might be of any type; for example, HTML, XML etc. CSS and JS are generally saved in static files.

2.2 Selecting Gateway

Several payment channels were considered, including PayTM, Razor Pay, and SBI ePay. While most of them ask for a high price from the user, it is crucial to emphasise that they must also provide excellent assistance and documentation. This would allow for more secure development and improved dispute resolution in the event of a botched recharge. Notably, two of these gateways, Cashfree and Zaakpay (MobiKwik), stood out for the reasons listed below and were chosen for the project.

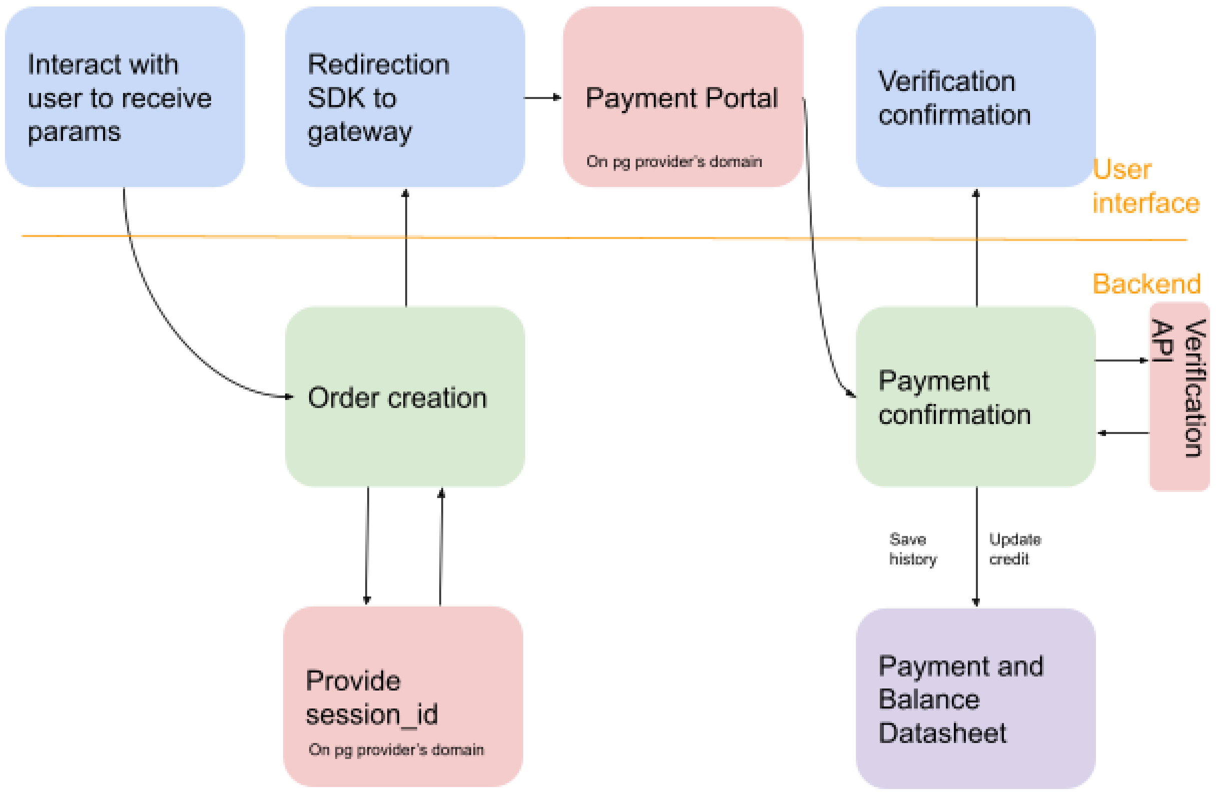
<u>Cashfree</u> is a well-known payment gateway provider that provides safe online transaction services. It accepts credit/debit cards, net banking, UPI, and digital wallets as payment methods. Cashfree is known for its 0% transaction fee on all UPI transactions, which eliminates the need for additional fees for such payments. Cashfree also does not charge annual or maintenance fees, assuring cost-effectiveness throughout the project.

<u>Mobikwik</u> is a well-known payment gateway noted for its simple UI and quick payment processing. It accepts credit/debit cards, net banking, UPI, and digital wallets as payment methods. MobiKwik's 0% transaction charge on UPI transactions adds to its appeal by lowering the costs associated with such payments. Furthermore, Mobikwik has no annual or maintenance costs, making it financially advantageous for the enterprise.

Section 3

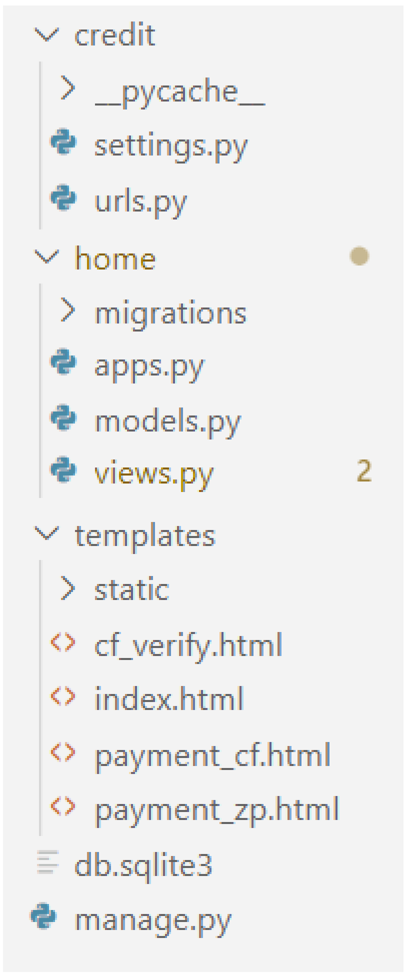
Procedure

3.1 Overview of Overall System



Described above is the entire workflow of the website.

The user supplies the portal with the roll number and the payment amount. It sends the form to the order creation and redirection page as a post request. The page sends a post request to the Cashfree/Mobikwik servers to create a new order for the amount of money and roll no (encoded in order_id), and the server responds with the session_id, which can be used to create the payment instance associated with that order.



This session_id is sent from the backend page to the front end page, which serves as a waiting page and loads the SDK required to send an encrypted request to the appropriate pg website, which then sends the user to the payment gateway.
When a user completes payment on the pg provider's website, the website is prompted by a redirecting (waiting) page to return to a specific URL with the order id as a get request. The backend then utilises the order id to confirm the payment, again using endpoint APIs with the corresponding pg provider. In response, the website sends payment confirmation to our backend, which recharges the wallet and displays a message to the user with the status of the current transaction and recharge confirmation. It also shows the user a history of all recharges to that specific roll no.
Given on the left is the basic structure of the project folder. (Files out of interest have been edited off)
Views, Templates and models have already been explained earlier. Along with these, we have got:
manage.py, which manages the admin, running of the server, updating files etc.
Db.sqlite3 is our database file managed through models
Setting.py contains the paths to respective files, setting universal environment and counting for created apps in Django.
Urls.py, on the other hand, will contain url patterns which shall be entered in the browser and map them to their respective views, which will return the files to be provided at those URLs to the client.

3.2 Landing Page

3.2.1 Frontend

We create an index.html in templates folder and create a simple view in view.py in home app created via manage.py startapp:

```
def home_view(request, *args, **kwargs):
    return render(request, "index.html", {})
```

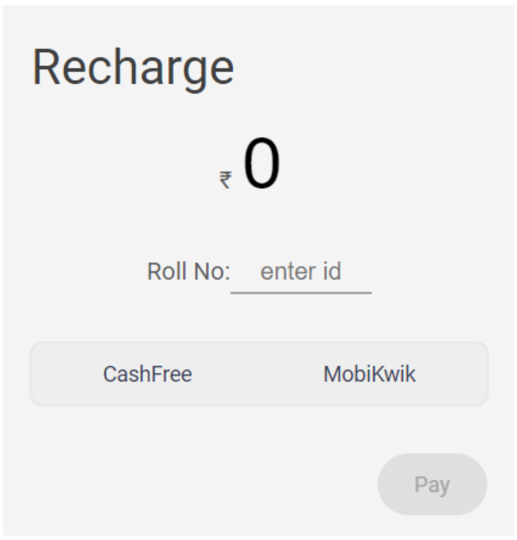
Home_view will return the index.html, which is automatically fetched from the templates by default. This is the most basic view, which returns an HTML file to be rendered with no server-end processing. {} stands for the content, i.e. the list of variables passed into index.html. In that case, however, it is empty, i.e. no content is passed.
This view is then mapped to the home address via urls.py as,

```
from django.contrib import admin
from django.urls import path
from home.views import home_view

urlpatterns = [
    path('', home_view, name='home'),
]
```

In order to test the website we can now access it through http://127.0.0.1:8000/ i.e is a local host that can be used to host a django server using manage.py runserver.

```
Python manage.py runserver
```

As we discussed, a view is a handler that accepts HTTP requests, processes them, and returns the HTTP response. It uses Models to retrieve the data needed to fulfil the request and Templates to render it on the user interface. Thus, whatever the view returns will be visible to the user.
The index.html file contains a simple form,



This form is basic HTML for inputting the amount for the recharge roll number for the card to be recharged and the choice of the portal. The aesthetic looks are due to styling using CSS.

An example of a basic form is below. It posts all the input data(only the amount here) to domain /cfrdr/ (the redirect server tagged order creation), decided by action.

```
<form method="POST" action="/cfrdr/"> {%csrf_token%}
        <input name="amt"type="number">
    </form>
```

One may notice the tag:

```
{%csrf_token%}
```

All the tags in {{ }} or {% %} are reserved for Django to provide an HTML page with content (variables passed in the view from Python).

The CSRF (Cross-Site Request Forgery protection) middleware and template tag offer simple protection against Cross-Site Request Forgeries. This type of attack occurs when a malicious website has a link, a form button, or some JavaScript that is meant to perform some action on your website using the credentials of a logged-in user who visits the malicious site in their browser.

Using javascript on the current page, we can limit the minimum and maximum amount allowed along with the correct roll number. To do so, we may simply limit the input amount via the following:

```
var inamt = document.getElementById('ib').value;
  inamt = inamt.replace(/^0+/, '');
  if (inamt.length == 0) {
    inamt = '0';
  } else if (parseInt(inamt) > 10000) {
    inamt = inamt.substring(0, inamt.length - 1);
  };
```

And limit input roll number to correct format via:

```
function rollcheck(str){
  var code, i, len;
  if (str.length<7){
    return false;
  }
  for (i = 0, len = str.length; i < len; i++) {
    code = str.charCodeAt(i);
    if((i == 0 || i == 1) && !(code > 47 && code < 58)){
      return false;
    }
    if((i == 2 || i == 3) && !(code > 64 && code < 91)){
      return false;
    }
    if((i == 4 || i == 5 || i == 6) && !(code > 47 && code < 58)){
      return false;
    }
  }
  return true;
};
```

This will make sure the roll number is in the format of two numbers followed by two alphabets followed by three numbers.

When three criteria are met, i.e., a gateway is chosen, the roll number is correct, and the amount is within the limit, we can simply enable the button. I.e. apply three if statements and enable the button when all three are satisfied.

3.3 Redirecting Page

3.3.1 Backend

Accepting Post Requests

We have to receive the post request from the landing page of /cfrdr/ (/zprdr/ for zaakpay) since, in order to create an order, we must have the amount and order_id, which is the roll number combined with the current date and time.

For cashfree

First we create the view as:

```
def cf_redirect_view(request, *args, **kwargs):
```

And assign it to cfrdr in urls.py.
Now we see the first argument of this function is request. It is the request needed, which could be post or get.
We read it using request.POST as,

```
def cf_redirect_view(request, *args, **kwargs):
    received = request.POST

*****your python code goes here*****

 return render(request, "payment_cf.html", my_cont)
```

Now we create another post request from within the python i.e writing the code of this view further,

```

    url = "https://sandbox.cashfree.com/pg/orders"

    payload = {
        "customer_details": {
            "customer_id": received.get('roll'),
            "customer_name": "abhay",
            "customer_email": "a@gmail.com",
            "customer_phone": "8888888888"
        },
        "order_id": str(received.get('roll')) + 'dt' +
dt.datetime.now().strftime("%d_%m_%Y_%H_%M_%S"),
        "order_meta": {"return_url":
"http://studentmess.iiserkol.ac.in/?order_id={order_id}"},
        "order_amount": str(received.get('amt')),
        "order_currency": "INR",
        "order_note": str(received.get('roll')) + ' dt ' +
dt.datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    }
    headers = {
        "accept": "application/json",
        "x-client-id": "TEST4181XXXXXXXXXXX",
        "x-client-secret": "TESTc8fbd80XXXXXXXXXXXXX",
        "x-api-version": "2022-09-01",
        "content-type": "application/json"
    }

    response = requests.post(url, json=payload, headers=headers)
```

We must import requests library to python to send post request.
Thus, the response to the request is returned in variable response in the last line.
Header contains respective secret client id and public client id provided by cashfree.

```
received.get('roll')
```

And similarly ‘amt’ both are received in the post request we saved in the ‘received’ variable earlier.
Payload is thus a json file of all information required for the required to create the order.
The response returns a long string, which can be converted to json using json.load() and contains ‘payment_session_id’ which is the only thing required for now, to redirect the user to the payment page.

<u>For mobikwik</u>

The processing is slightly different but everything else is same
We define some parameters just as in cashfree

```
params = {"amount": amount, "merchantIdentifier": merchantIdentifier,
                  "orderId": orderId, "currency": currency,
"buyerEmail": buyerEmail}
```

And process the transaction using

```
def processTransaction():
        requestParams = params
        requestUrl = Config.ENVIRONMENT + Config.TRANSACTION_API_URL
        checksumString = getChecksumString(requestParams)
        checksum = calculateChecksum(ZAAKPAY_SECRET_KEY,
checksumString)
return(requestUrl+"?"+checksumString+"checksum="+checksum)
```

The only thing different here is that order creation directly create the request by encrypting the checksum string which is nothing but the parameters, joined together with & just as in get requests. This checksum is integrated using calculateChecksum function which is as follows:

```
    def calculateChecksum(secret_key, checksumString):
        secret_key = bytes(secret_key, "utf-8")
        total_params = bytes(checksumString, "utf-8")
        checksum = hmac.new(secret_key, total_params,
                            hashlib.sha256).hexdigest()
        return checksum
```

Uses an HMAC SHA-256 algorithm to calculate the checksum of the data passed.

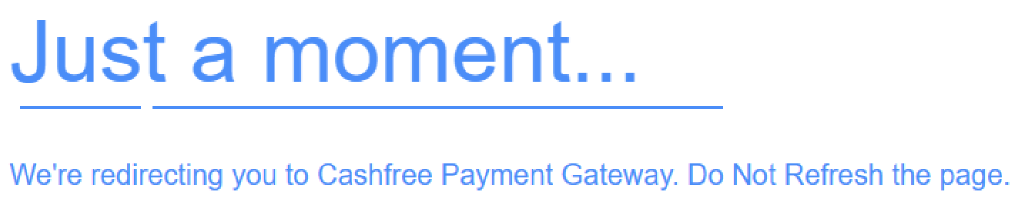
This final link generated by processTransaction can be used directly using a simple post request with request parameters given in the function. You need to post all the parameters with a checksum to the request URL, and you can skip the frontend step, unlike cashfree, which uses a cashfree SDK on the redirect page.

3.3.2 Frontend

Redirect frontend is nothing but simply the payment_cf.html in templates returned by /cfrdr/ while we get the request, from backend and return it to front end by passing it as a content (in a list in json form). So, we pass the session_id and the order_id in the content and extract them in payment_cf.html using {{ var }} notion.

Fundamental warning

We give the client a simple, prominent warning not to close the tabs at any point throughout the transaction. This is to avoid ending up in a situation where the user has paid, but cashfree does not redirect to the verification page on our server, which we shall discuss later. If any such issue happens, we will not be informed of payment, and the mess card will not be updated. However, money may be deducted.


Just a moment...

We're redirecting you to Cashfree Payment Gateway. Do Not Refresh the page.

This page features an amazing use of CSS @keyframe method for loading bar animation.

```
@keyframes "loading" {
     from { left: 0; }
     to { left: 400px; }
     }
```
And applying
```
-webkit-animation: loading 2s infinite;
```
Three dots were placed on that blue line. Those three dots move from left to right due to this animation, causing this loading bar effect without javascript. Meanwhile, cashfree SDK loads and executes redirection in the background.

Cashfree SDK

Cashfree SDK is loaded from an external script from their website
```
<script src="https://sdk.cashfree.com/js/v3/cashfree.js"></script>
```
```
    let checkoutOptions = {
      paymentSessionId: '{{ psid }}',
      returnUrl: "http://127.0.0.1:8000/verify/?orderid={{ oid }}",
    };
    const cashfree = Cashfree({
      mode: "sandbox" //or production
    });

    cashfree.checkout(checkoutOptions)
```

The checkout function, upon execution, will redirect to the cash free payment gateway. However, as discussed earlier, it requires session_id provided by content, via {{ psid }} passed into the render via cf_redirect_view.
Return URL tells cashfree to return to the verification page after the transaction. {{ oid }} is order_id parameter.

A frontend redirect page is not required for Mobikwik.

3.4 Gateway

Client has been redirected to cashfree or mobikwik Payment Gateway and pays the required sum or cancels the payment request. In either case , the client will be redirected to verificationPage
http://127.0.0.1:8000/verify/?orderid={{ oid }}.
The page will give a get request of order id to the verification page.

3.5 Verification

3.5.1 Api Endpoint For Payment Verification

Whatever the user's actions are on the payment gateway are returned by the gateway as a response to APIs provided by both Zaakpay and Cashfree.

Cashfree

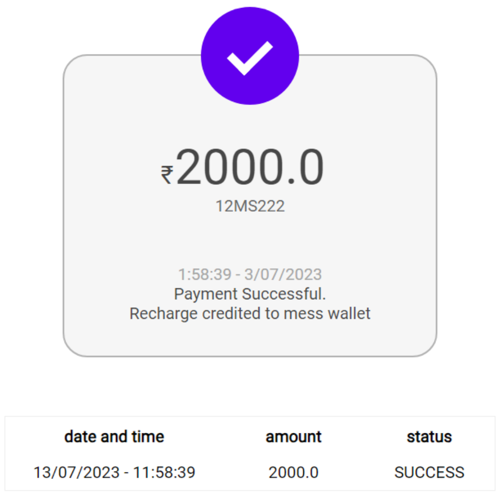
```
def cf_verify_view(request, *args, **kwargs):
    rcd = request.GET
    url = "https://sandbox.cashfree.com/pg/orders/" + \
        rcd.get('orderid') + "/payments"
    headers = {
        "accept": "application/json",
        "x-client-id": "TEST4181XXXXXXXXX",
        "x-client-secret":"TESTc8fXXXXXXXX",
        "x-api-version": "2022-09-01"
    }
    response = requests.get(url, headers=headers)
    print(response.text)
    my_cont = {'oid': order id,
               'amnt': amount,
               'sts': status,
               'hst': history to display
               }

    return render(request, "cf_verify.html", my_cont)
```

Again the view is mapped to /verify/ in urls.py. One can see, that verification of payment for cashfree is a simple POST request providing three things,

- Order id
- Client public key
- Client private key

And in return gets another json response. We store following from the json in our database :

- Order_amount
- Order_id
- payment_status



₹2000.0
12MS222

1:58:39 - 3/07/2023
Payment Successful.
Recharge credited to mess wallet

date and time	amount	status
13/07/2023 - 11:58:39	2000.0	SUCCESS

and also return them to the rendered page to give the user a response sheet confirming their payment and mess recharge status immediately. Which is displayed through basic html front end.

The image shows the confirmation page for a successful payment. It Also displays the table of all previous transactions done by that roll no. pass into it as {{ hst }} , which we will discuss ahead how.

<u>Mobikwik</u>

Almost same parameters are asked for by Zaakpay,

- Checksum (needs to be recalculated)
- Order_id
- merchent_id

And returns a json with ‘payment_status’ to verify the status of the particular transaction.

```
url = "https://zaakstaging.zaakpay.com/checktransaction?v=5"

payload = {
    "mode": "0",
    "merchantIdentifier": "mercent_id",
    "orderId": "order_id",
    "checksum": "checksum"
}
headers = {
    "accept": "application/json",
    "content-type": "application/x-www-form-urlencoded"
}

response = requests.post(url, data=payload, headers=headers)

print(response.text)
```

Just the same as cashfree.

3.5.2 Database Management

The next aim is to save the order ID amounts and payment status in a database to keep track of the current balance and to prevent clashes in case the verification page is reloaded or payment is not received but is made on the gateway by the client.

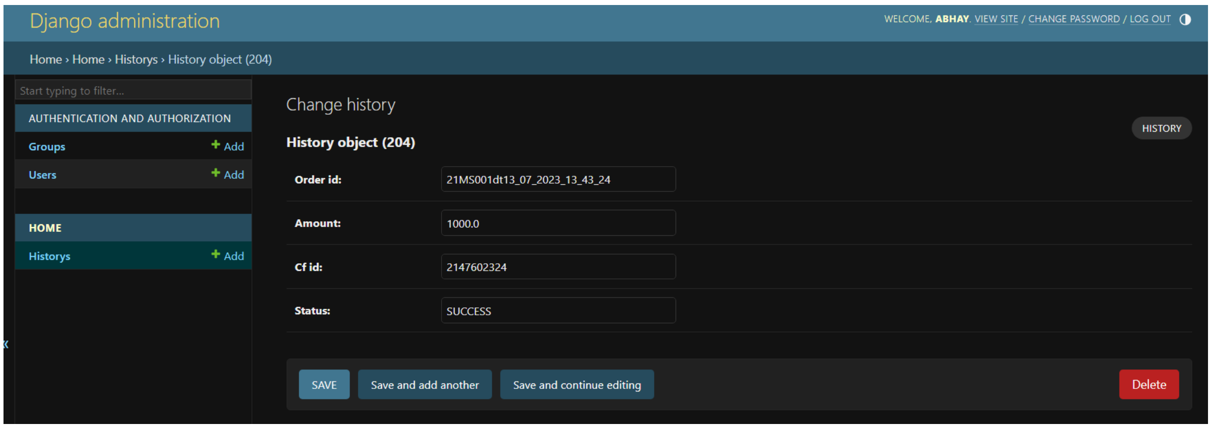
For this, we use models.py. Remember, we discussed earlier that a Model is an object that defines the data structure of a Django application. It is in charge of keeping the entire application's data and provides numerous techniques for adding, updating, reading, and deleting data from the database. So, we create a new model calling it History, in models.py:

```
from django.db import models

class History(models.Model):
  order_id = models.CharField(max_length=255)
  amount  = models.CharField(max_length=10)
  cf_id = models.CharField(max_length=255)
  status  = models.CharField(max_length=10)
```

Make sure to migrate the new models to server using,

```
py manage.py makemigrations home
py manage.py migrate
```

Migrations are Django's method of propagating model changes (adding a field, deleting a model, etc.) into your database schema. They are intended to be mostly automatic, but you will need to know when to make migrations when to run them, and how to deal with frequent issues.


Django administration
WELCOME, ABHAY. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Home › Historys › History object (204)
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
HOME
Historys + Add
HISTORY
SAVE
Save and add another
Save and continue editing
Delete

Change history

History object (204)

Order id:	21MS001dt13_07_2023_13_43_24
Amount:	1000.0
Cf id:	2147602324
Status:	SUCCESS

The image above shows the Django admin page which is accessible through http://127.0.0.1:8000/admin/ via a superuser. To create admin user:

```
py manage.py createsuperuser
```

Will guide one through all the steps to set up admin. Admin panel will reflect all the models created in django, Like here is an example of our successful history model.

3.5.3 Crediting To User’s Mess Card

To read to models, we use model.object.filter/all.values()
To write to models, we use model.object.create(params)

```
from .models import History
if(History.objects.filter(order_id=oid).exists() == False):
   History.objects.create(order_id=oid, amount=amnt,
                                   cf_id=cfid, status=sts)
   ***insert to original database of mess credits, the new balance for
the roll number.

hst = History.objects.filter(order_id__startswith=oid[0:7]).values()
```

History.objects.filter(order_id=oid).exists() checks for pre-existing same order id; do not double the entry. We must do so, or refreshing the page would lead to double credit to the user's mess card.
Remember, hst, it was the variable passed to cf_verify.html. It contains the list of all objects of the History model, order_id of which starts with the roll number of the client (first seven characters of order_id list). This list will also be printed on the verification response.

Section 4

Conclusion

4.1 Summary

- The project aimed to learn how to create a frontend web page using HTML, CSS, and JS and integrate a payment gateway, enabling wallet top-up capabilities.
- The project also aimed to learn how to stand up to industrial security standards for APIs and requests backend(Django) for such sensitive data.
- The project's goals were to evaluate the payment gateway integration's usability, functionality, and affordability, gain knowledge in integrating payment gateways and web development, design a responsive, functional, and intuitive user interface, automate updating wallet balances, and integrate the payment gateway.
- The project successfully integrated the payment gateway and created a functional and user-friendly frontend web page.
- The project faced some challenges, such as errors in the payment gateway integration and difficulty styling the frontend web page.
- The project learned several skills: web development, payment gateway integration, and error handling.

The Cashfree and Zaakpay payment gateway were utilised for the project, which are well-known and trustworthy payment processors in India.
The frontend web page was created to be responsive, so it would appear excellent on all platforms, including desktop computers, laptop computers, tablets, and smartphones.
The project employed various JavaScript approaches to make the frontend web page more dynamic and user-friendly.
Django was used backend to securely communicate with the frontend and Cashfree servers while keeping private keys secure.
The project used Django's error-handling techniques to ensure the frontend web page would continue functioning even if the payment gateway integration failed.
Overall, the initiative intends to improve the student experience by providing an easy and effective means to recharge wallets and boosting administrative effectiveness and transaction management. It sets the way for new advancements and demonstrates how adopting technology may enhance operations and customer happiness.

4.2 Skills Acquired

- Web development: Having a solid understanding of HTML, CSS, and JavaScript, as well as the capacity to add interactive components and build well-structured websites.
- Secure integration: The project learned how to integrate a payment gateway into a web page, and hence When dealing with sensitive data, such as payment information, it is essential to be aware of appropriate security standards.
- Error handling: The project learned how to handle errors in the payment gateway integration.Familiarity with browser developer tools, logging mechanisms, and code review practices to aid problem-solving and, most importantly, security measures.
- Responsive design: The project learned to design a front-end web page that looks good on all devices. Knowledge of responsive design strategies allows for creating web pages that adapt to multiple screen sizes and devices effortlessly.
- Django and Database(SQLite3): Backend development expertise using Django, a robust Python web platform. Modelling abilities, database operations, and using Django's ORM for fast data administration and retrieval.
- JavaScript techniques: The project learned to use JavaScript to make a more interactive and user-friendly front-end web page.

In addition to these technical skills, the project also learned several soft skills, such as:

- Problem-solving: The project had to solve several problems to integrate the payment gateway and create a functional front-end web page.
- Communication: The project had to communicate with the project's supervisor and other team members to get feedback and resolve issues.
- Time management: The project had to manage its time effectively to complete it on time.

Section 5

Challenges and Ways to Overcome

- Learning Django: Learning Django from scratch in a limited period is hard, but not highly. The language and framework are very straightforward once referred to tutorials and proper documentation provided by freecodecamp and w3schools.
- Responsive Design: Designing for different browsers to render the page similarly is challenging. Several components do not render the same on Firefox. Designing the webpage as responsive and adaptable to different screen sizes and devices presented a challenge. Ensuring proper layout, font sizes, and image scaling for various devices was challenging.
- User Experience Optimization: Creating an aesthetically pleasing and user-friendly design was challenging while balancing form validation, error handling, and clear instructions.
- Payment Gateway Integration: Due to the necessity to manage secure communication, transaction processing, and error management, integrating the chosen payment gateway (such as Cashfree or Mobikwik) was undoubtedly a challenging undertaking. Becoming familiar with the payment gateway's API documentation and guaranteeing correct integration was challenging.
- Testing and Debugging: It took much effort to conduct thorough testing to find and fix functional, compatibility, or usability problems. During the development and testing phases, issues needed to be carefully analysed and problems needed to be solved.

Regarding the final point, Some fixes required to be done throughout the project are,

- Tinkering how to set up a primary Django server and run the first view while creating the start app, defining paths in settings.py and assigning urls.py URLs to specific views.
- Getting to know strict industrial-grade security standards while trying to request the pg servers for order creating and getting continuous errors, requests being rejected for one thing or another.
- Struggling with unupdated migrations and more importantly unupdated static files. Initial original static files were being edited and then `python manage.py collectstatic --noinput --clear` Was run causing old files to replace new ones which were elsewhere. Such issues are very common while working with any coding lang.

- Another error was encountered when the returned (response) JSON by cashfree after verification of the done transaction returned a list of JSONs and went unnoticed. To access the JSON, we must refer to the 0th element of the list to look at the last transaction, which possibly would be the successful one, if there is any.

  ```
  json_object = json.loads(response.text)
      print(json_object[0])
  sts = json_object[0].get('payment_status')
  ```

  Is the syntax to fetch a particular element for a given response. Other JSONs are previously done transactions in the same session, which might have failed and thus did not redirect.

Provided solutions, along with Dr Dwaipayan Roy's guidance, were effective enough to allow the project to overcome these challenges and many other challenges that were avoided before they could occur.