Test Document

for

PedalPal

Version 1.0

Prepared by

Group # 4 **Group Name: Bit Brewers**

Raghav Manglik	220854	raghavkmanglik@gmail.com
Amogh Bhagwat	220288	amogh.2004b@gmail.com
Srishti Chandra	221088	chandra.srishti2403@gmail.com
Wadkar Srujan Nitin	221212	srujanwadkar@gmail.com
Anaswar K B	220138	anaswarkb013@gmail.com
Khushi Gupta	220531	khushi07g@gmail.com
Ananya Singh Baghel	220136	ananyabaghel2004@gmail.com
Pathe Nevish Ashok	220757	nevu.pathe1234@gmail.com
Debraj Karmakar	220329	debraj2003jsr@gmail.com
Kaneez Fatima	220496	kaneezfatimamehdi7@gmail.com

Course:	CS253
Mentor TA:	Mr. Bharat
Instructor:	Prof. Indranil Saha
Date:	January 25, 2024

Contents

1. Revisions

Version	Primary Author(s)	Description of Version	Date Completed
v1.0	Raghav Manglik Amogh Bhagwat Srishti Chandra Wadkar Srujan Nitin Pathe Nevish Ashok Debraj Karmakar Khushi Gupta Ananya Baghel Anaswar K B Kaneez Fatima	First version of the Test Document	29/03/24

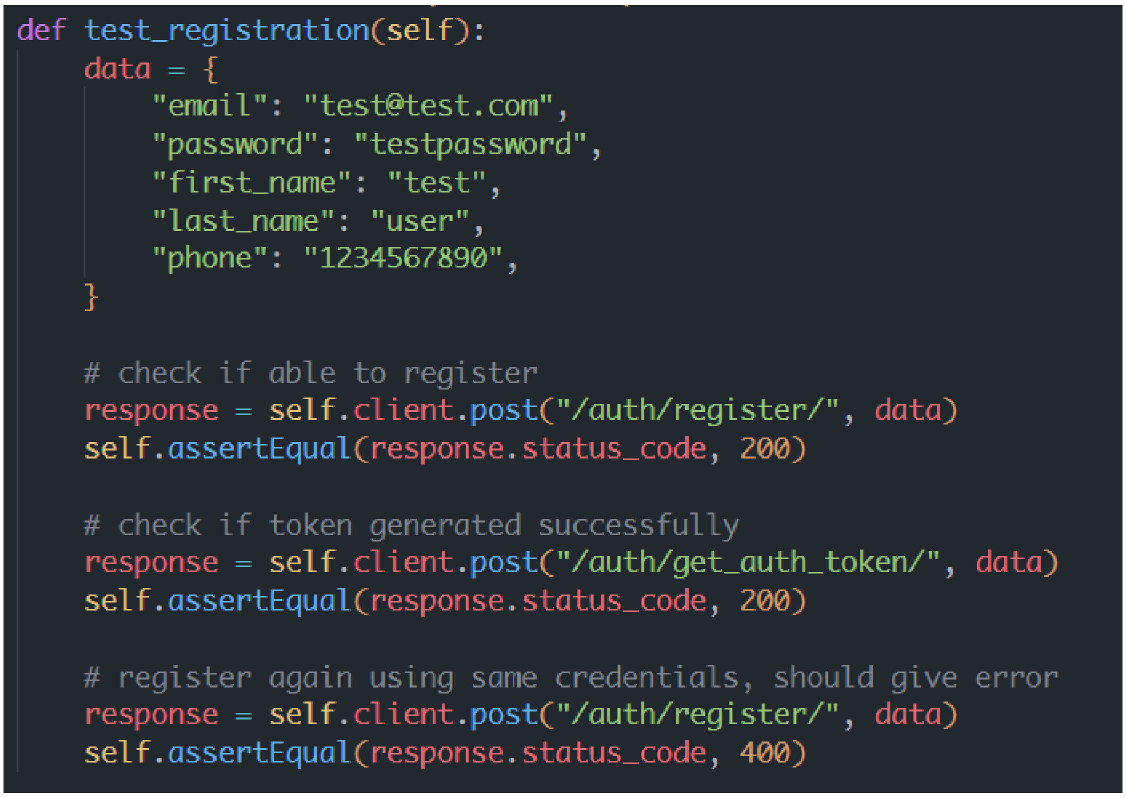
2. Introduction

3. Unit Testing

3.1 Authentication

3.1.1 Registering a User

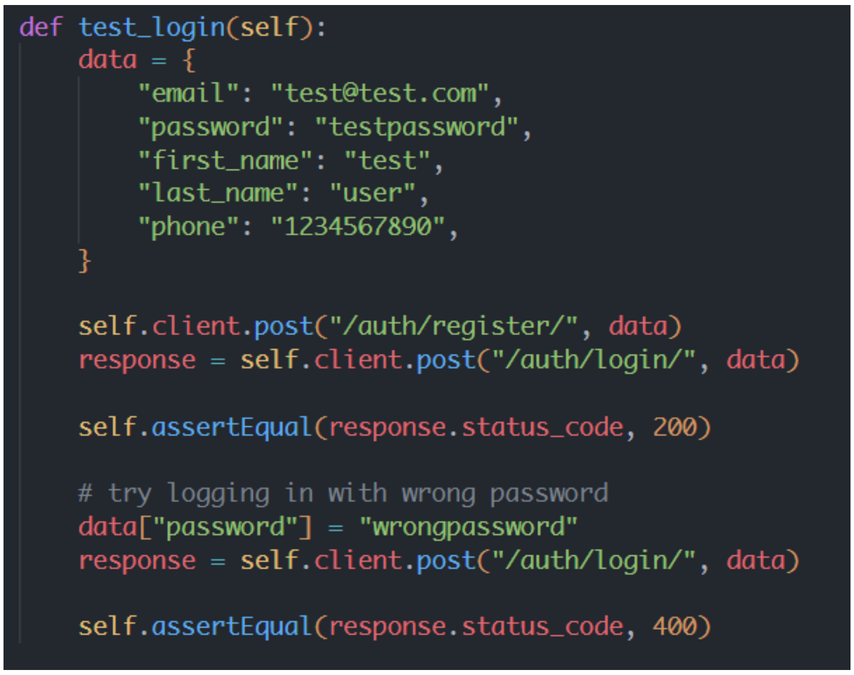
API Endpoint: `/auth/register/`
Test Owner: Amogh Bhagwat
Date: 25/03/2024
Test Description: This test case is used to check if a user can register successfully.
Test Results: User is able to register successfully. If a user with the email already exists, appropriate error message is shown. Authentication token is also generated correctly.



```
def test_registration(self):
    data = {
        "email": "test@test.com",
        "password": "testpassword",
        "first_name": "test",
        "last_name": "user",
        "phone": "1234567890",
    }

    # check if able to register
    response = self.client.post("/auth/register/", data)
    self.assertEqual(response.status_code, 200)

    # check if token generated successfully
    response = self.client.post("/auth/get_auth_token/", data)
    self.assertEqual(response.status_code, 200)

    # register again using same credentials, should give error
    response = self.client.post("/auth/register/", data)
    self.assertEqual(response.status_code, 400)
```

3.1.2 Logging in a User

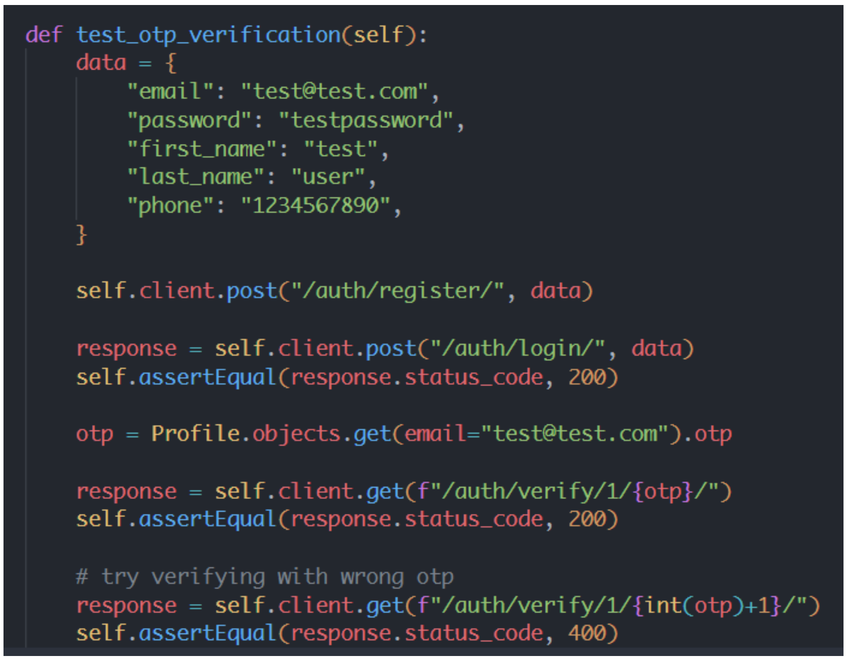
API Endpoint: `/auth/login/`
Test Owner: Amogh Bhagwat
Date: 25/03/2024
Test Description: This test case is used to check if a user can login successfully.
Test Results: User is able to login successfully. If the user enters wrong password or the user does not exist, appropriate error message is shown.



```python
def test_login(self):
    data = {
        "email": "test@test.com",
        "password": "testpassword",
        "first_name": "test",
        "last_name": "user",
        "phone": "1234567890",
    }

    self.client.post("/auth/register/", data)
    response = self.client.post("/auth/login/", data)

    self.assertEqual(response.status_code, 200)

    # try logging in with wrong password
    data["password"] = "wrongpassword"
    response = self.client.post("/auth/login/", data)

    self.assertEqual(response.status_code, 400)
```

3.1.3 Email Verification

API Endpoint: `/auth/verify/`
Test Owner: Amogh Bhagwat
Date: 25/03/2024
Test Description: This test case is used to check if a user can verify their email successfully.
Test Results: User receives the mail containing the link to verify their email successfully. On clicking the link, their email is verified. If the OTP in the link is modified, the verification fails.



```python
def test_otp_verification(self):
    data = {
        "email": "test@test.com",
        "password": "testpassword",
        "first_name": "test",
        "last_name": "user",
        "phone": "1234567890",
    }

    self.client.post("/auth/register/", data)

    response = self.client.post("/auth/login/", data)
    self.assertEqual(response.status_code, 200)

    otp = Profile.objects.get(email="test@test.com").otp

    response = self.client.get(f"/auth/verify/1/{otp}/")
    self.assertEqual(response.status_code, 200)

    # try verifying with wrong otp
    response = self.client.get(f"/auth/verify/1/{int(otp)+1}/")
    self.assertEqual(response.status_code, 400)
```

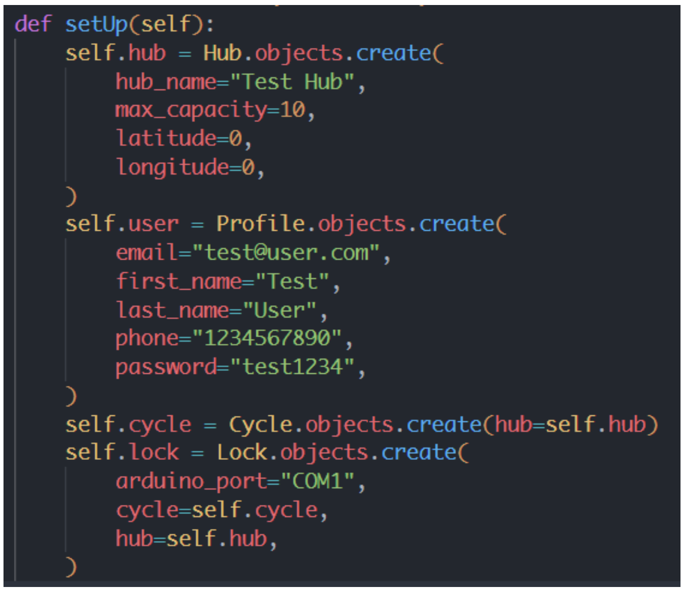
3.2 Cycle Bookings

3.2.1 Starting Ride Instantly

API Endpoint: `/booking/book/`
Test Owner: Khushi Gupta
Date: 26/03/2024
Test Description: This test case is used to check if a user can start a ride instantly by scanning the QR code on the lock.
Test Results: User is able to start the ride instantly by scanning the QR code on the lock. The following edge cases were tested -

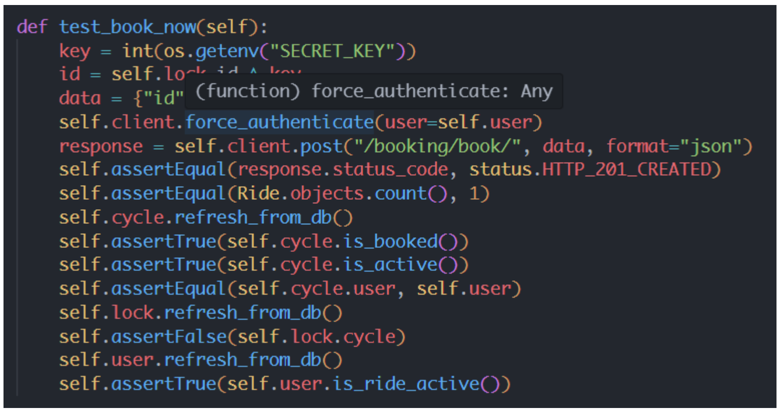
- If the user has negative balance in their wallet, they are denied to start the ride.
- If the QR code is invalid, appropriate error message is shown.
- If the lock has no cycle attached to it, appropriate message is shown.
- If the user already has an active ride in progress, they are denied to start another ride.
- If the cycle is already booked by some other user, appropriate message is shown.

The test database is initialized as follows

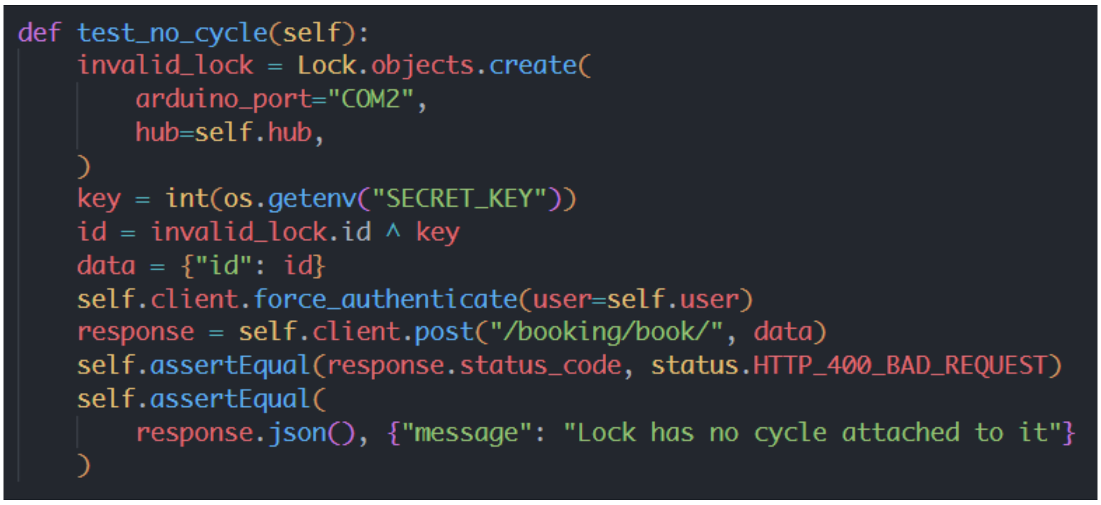


```python
def setUp(self):
    self.hub = Hub.objects.create(
        hub_name="Test Hub",
        max_capacity=10,
        latitude=0,
        longitude=0,
    )
    self.user = Profile.objects.create(
        email="test@user.com",
        first_name="Test",
        last_name="User",
        phone="1234567890",
        password="test1234",
    )
    self.cycle = Cycle.objects.create(hub=self.hub)
    self.lock = Lock.objects.create(
        arduino_port="COM1",
        cycle=self.cycle,
        hub=self.hub,
    )
```

The edge cases are tested as follows

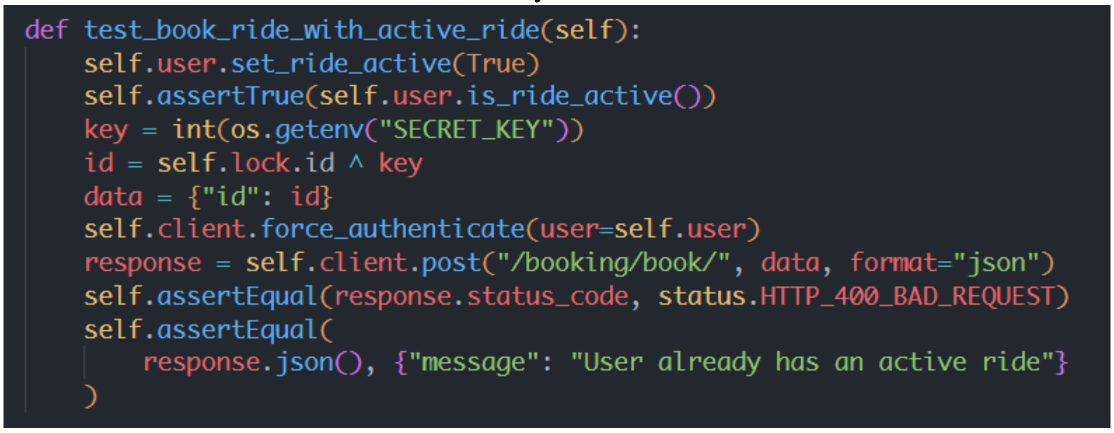


```python
def test_book_now(self):
    key = int(os.getenv("SECRET_KEY"))
    id = self.lock.id ^ key
    data = {"id" (function) force_authenticate: Any
    self.client.force_authenticate(user=self.user)
    response = self.client.post("/booking/book/", data, format="json")
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    self.assertEqual(Ride.objects.count(), 1)
    self.cycle.refresh_from_db()
    self.assertTrue(self.cycle.is_booked())
    self.assertTrue(self.cycle.is_active())
    self.assertEqual(self.cycle.user, self.user)
    self.lock.refresh_from_db()
    self.assertFalse(self.lock.cycle)
    self.user.refresh_from_db()
    self.assertTrue(self.user.is_ride_active())
```



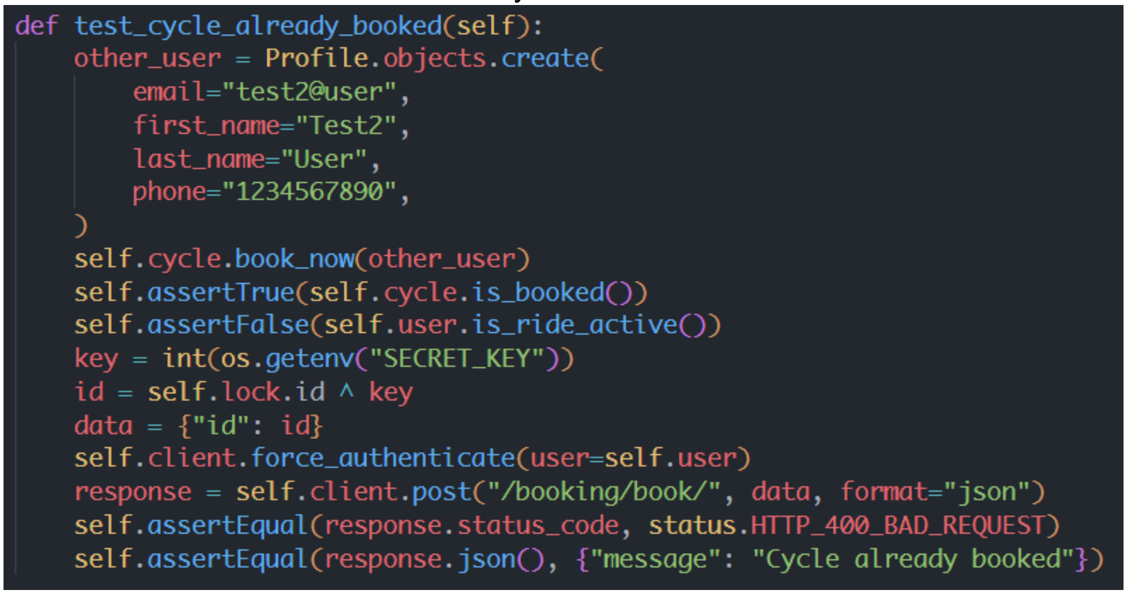
```python
def test_no_cycle(self):
    invalid_lock = Lock.objects.create(
        arduino_port="COM2",
        hub=self.hub,
    )
    key = int(os.getenv("SECRET_KEY"))
    id = invalid_lock.id ^ key
    data = {"id": id}
    self.client.force_authenticate(user=self.user)
    response = self.client.post("/booking/book/", data)
    self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
    self.assertEqual(
        response.json(), {"message": "Lock has no cycle attached to it"}
    )
```

Lock has no cycle attached to it



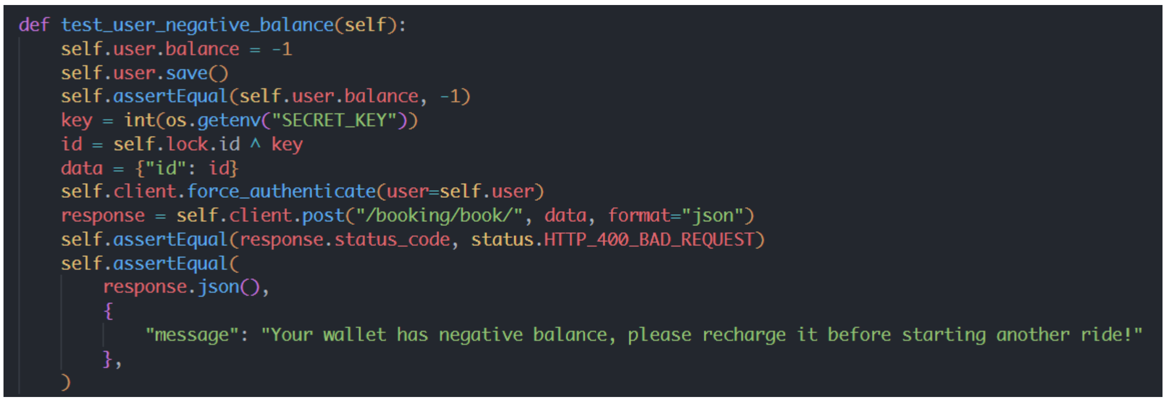
```python
def test_book_ride_with_active_ride(self):
    self.user.set_ride_active(True)
    self.assertTrue(self.user.is_ride_active())
    key = int(os.getenv("SECRET_KEY"))
    id = self.lock.id ^ key
    data = {"id": id}
    self.client.force_authenticate(user=self.user)
    response = self.client.post("/booking/book/", data, format="json")
    self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
    self.assertEqual(
        response.json(), {"message": "User already has an active ride"}
    )
```

User already has an active ride



```python
def test_cycle_already_booked(self):
    other_user = Profile.objects.create(
        email="test2@user",
        first_name="Test2",
        last_name="User",
        phone="1234567890",
    )
    self.cycle.book_now(other_user)
    self.assertTrue(self.cycle.is_booked())
    self.assertFalse(self.user.is_ride_active())
    key = int(os.getenv("SECRET_KEY"))
    id = self.lock.id ^ key
    data = {"id": id}
    self.client.force_authenticate(user=self.user)
    response = self.client.post("/booking/book/", data, format="json")
    self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
    self.assertEqual(response.json(), {"message": "Cycle already booked"})
```

Cycle is already booked by another user



```python
def test_user_negative_balance(self):
    self.user.balance = -1
    self.user.save()
    self.assertEqual(self.user.balance, -1)
    key = int(os.getenv("SECRET_KEY"))
    id = self.lock.id ^ key
    data = {"id": id}
    self.client.force_authenticate(user=self.user)
    response = self.client.post("/booking/book/", data, format="json")
    self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
    self.assertEqual(
        response.json(),
        {
            "message": "Your wallet has negative balance, please recharge it before starting another ride!"
        },
    )
```

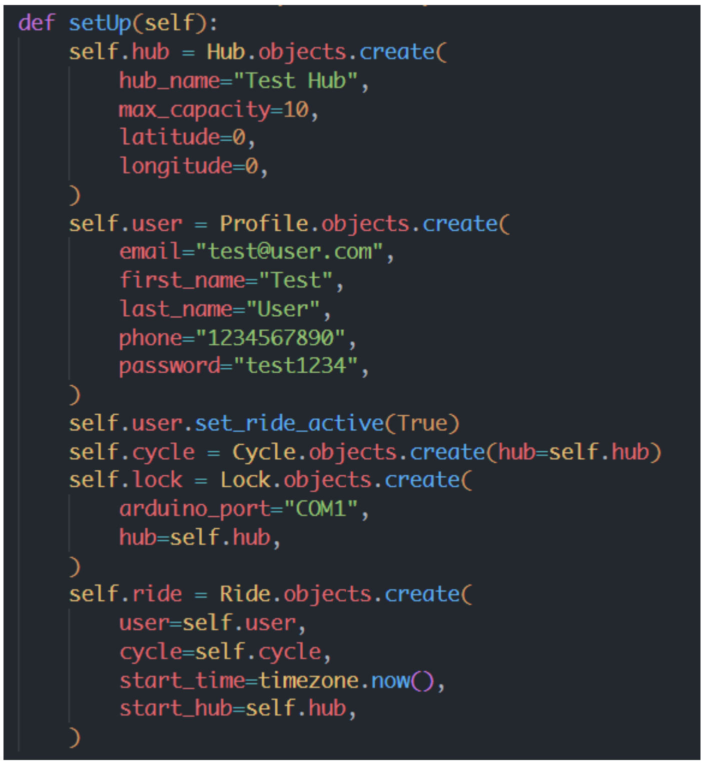
User has negative balance in their wallet

3.2.2 Ending Ride

API Endpoint: `/booking/end/`
Test Owner: Khushi Gupta
Date: 26/03/2024
Test Description: This test case is used to check if a user can end a ride successfully.
Test Results: User is able to end the ride successfully by scanning the QR code on the lock.
The following edge cases were tested -

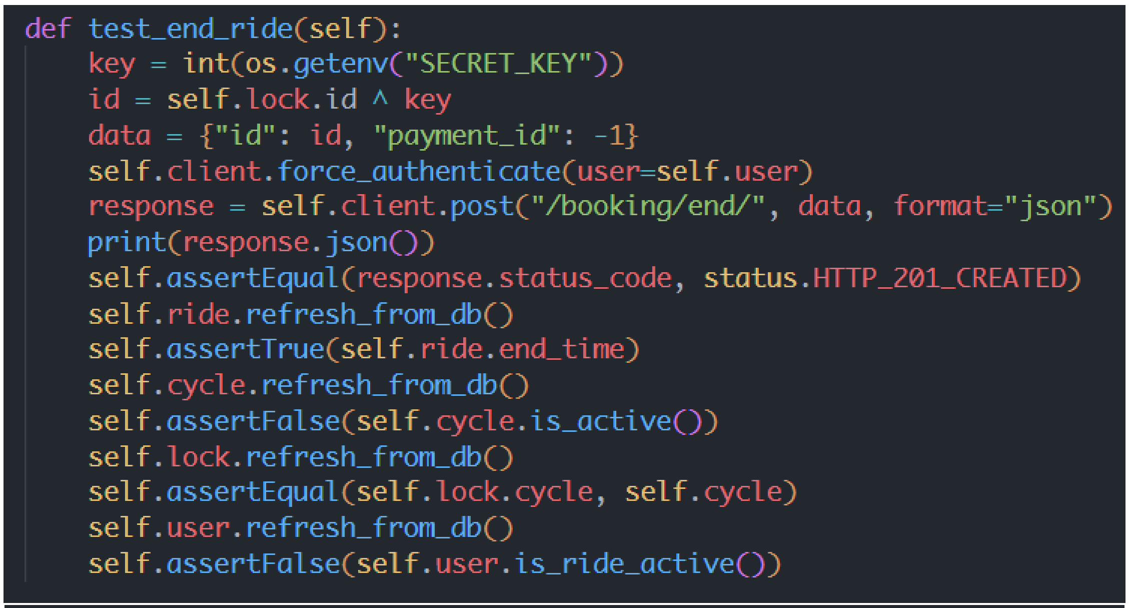
- If the QR code is invalid, appropriate error message is shown.
- If the lock is already attached to another cycle, appropriate message is shown.
- If the user does not have an active ride, appropriate message is shown.

The test database is initialized as follows

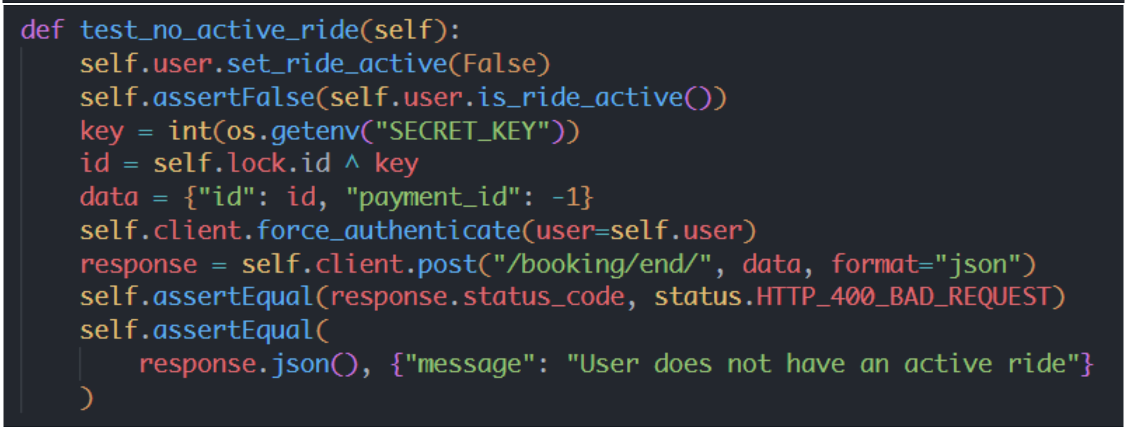


```python
def setUp(self):
    self.hub = Hub.objects.create(
        hub_name="Test Hub",
        max_capacity=10,
        latitude=0,
        longitude=0,
    )
    self.user = Profile.objects.create(
        email="test@user.com",
        first_name="Test",
        last_name="User",
        phone="1234567890",
        password="test1234",
    )
    self.user.set_ride_active(True)
    self.cycle = Cycle.objects.create(hub=self.hub)
    self.lock = Lock.objects.create(
        arduino_port="COM1",
        hub=self.hub,
    )
    self.ride = Ride.objects.create(
        user=self.user,
        cycle=self.cycle,
        start_time=timezone.now(),
        start_hub=self.hub,
    )
```

The edge cases are tested as follows

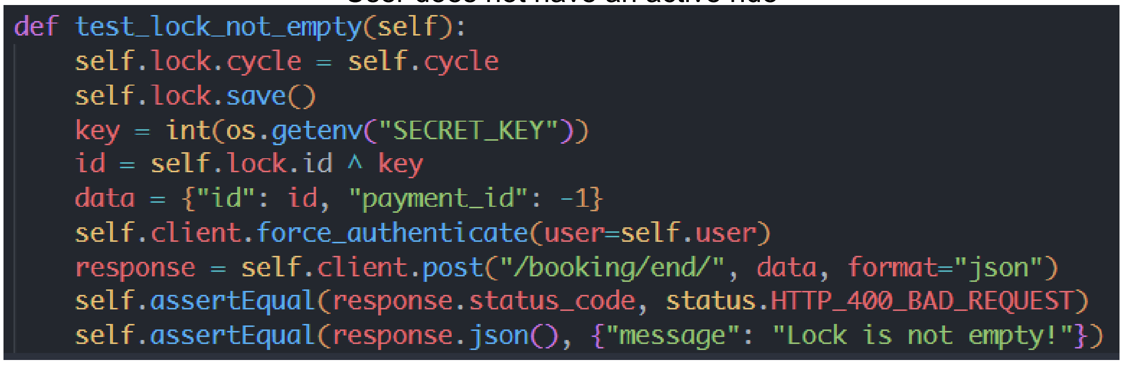


```python
def test_end_ride(self):
    key = int(os.getenv("SECRET_KEY"))
    id = self.lock.id ^ key
    data = {"id": id, "payment_id": -1}
    self.client.force_authenticate(user=self.user)
    response = self.client.post("/booking/end/", data, format="json")
    print(response.json())
    self.assertEqual(response.status_code, status.HTTP_201_CREATED)
    self.ride.refresh_from_db()
    self.assertTrue(self.ride.end_time)
    self.cycle.refresh_from_db()
    self.assertFalse(self.cycle.is_active())
    self.lock.refresh_from_db()
    self.assertEqual(self.lock.cycle, self.cycle)
    self.user.refresh_from_db()
    self.assertFalse(self.user.is_ride_active())
```



```python
def test_no_active_ride(self):
    self.user.set_ride_active(False)
    self.assertFalse(self.user.is_ride_active())
    key = int(os.getenv("SECRET_KEY"))
    id = self.lock.id ^ key
    data = {"id": id, "payment_id": -1}
    self.client.force_authenticate(user=self.user)
    response = self.client.post("/booking/end/", data, format="json")
    self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
    self.assertEqual(
        response.json(), {"message": "User does not have an active ride"}
    )
```

User does not have an active ride



```python
def test_lock_not_empty(self):
    self.lock.cycle = self.cycle
    self.lock.save()
    key = int(os.getenv("SECRET_KEY"))
    id = self.lock.id ^ key
    data = {"id": id, "payment_id": -1}
    self.client.force_authenticate(user=self.user)
    response = self.client.post("/booking/end/", data, format="json")
    self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
    self.assertEqual(response.json(), {"message": "Lock is not empty!"})
```

Lock is already attached to another cycle

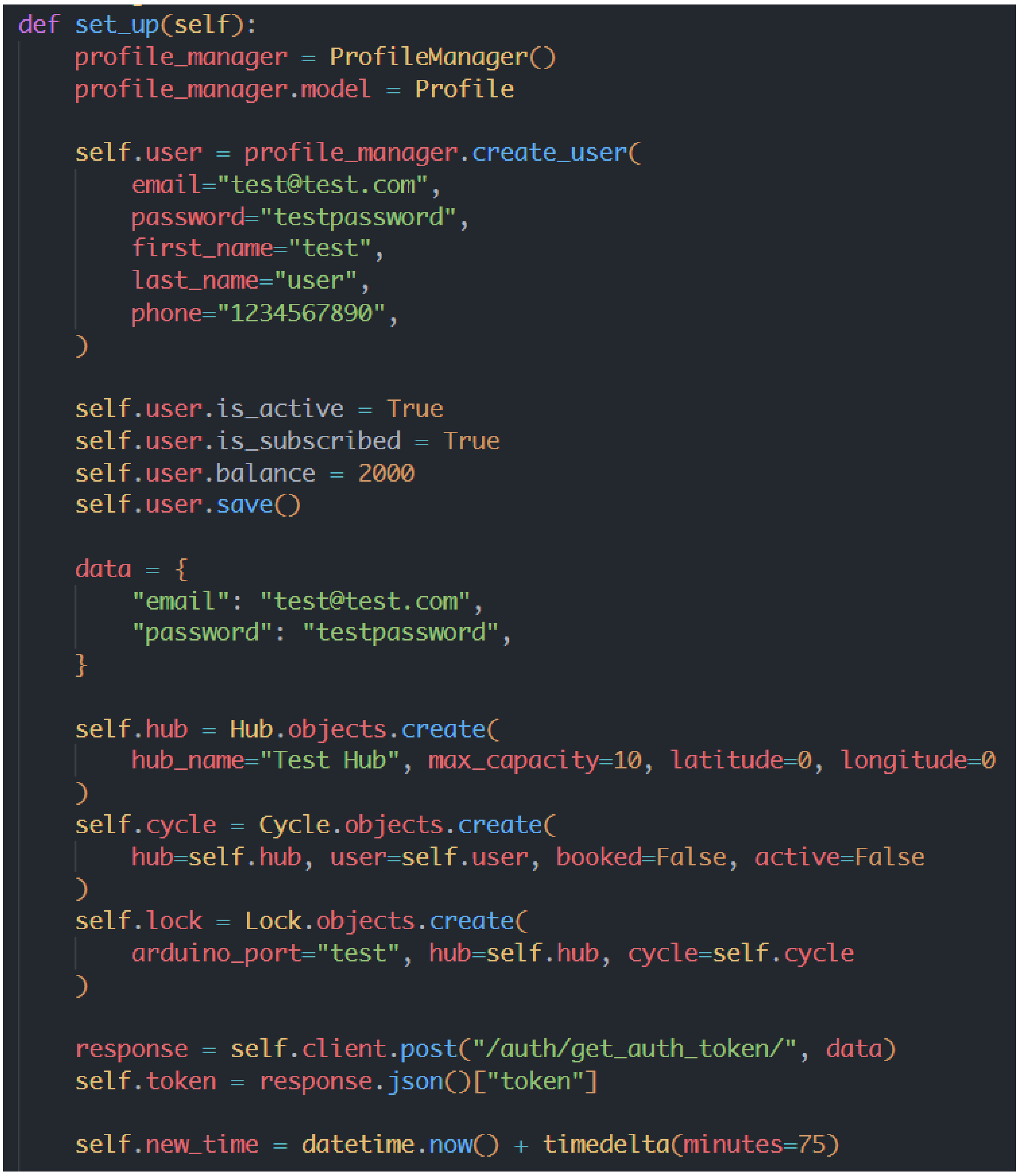
3.2.3 Booking a Cycle for Later

API Endpoint: `/booking/book_later/`
Test Owner: Nevish Pathe
Date: 26/03/2024
Test Description: This test case is used to check if a subscribed user can book a cycle for

later.
Test Results: User is able to book a cycle for later successfully. The following edge cases were tested -

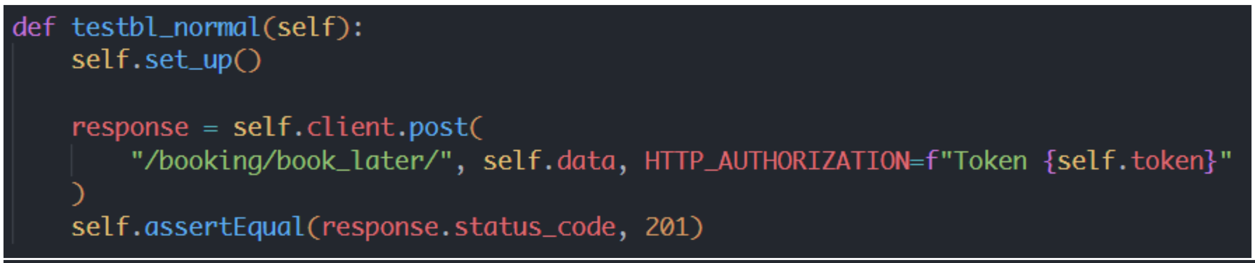
- If the user has insuffienct balance in their wallet, they are denied to book a cycle.
- If the user has not subscribed to the service, they are denied to book a cycle for later.
- If the desired cycle is already booked by some other user, appropriate message is shown.
- If the desired start time is in the past, appropriate message is shown.
- If there is no cycle available at the desired hub, appropriate message is shown.

The test database is initialized as follows

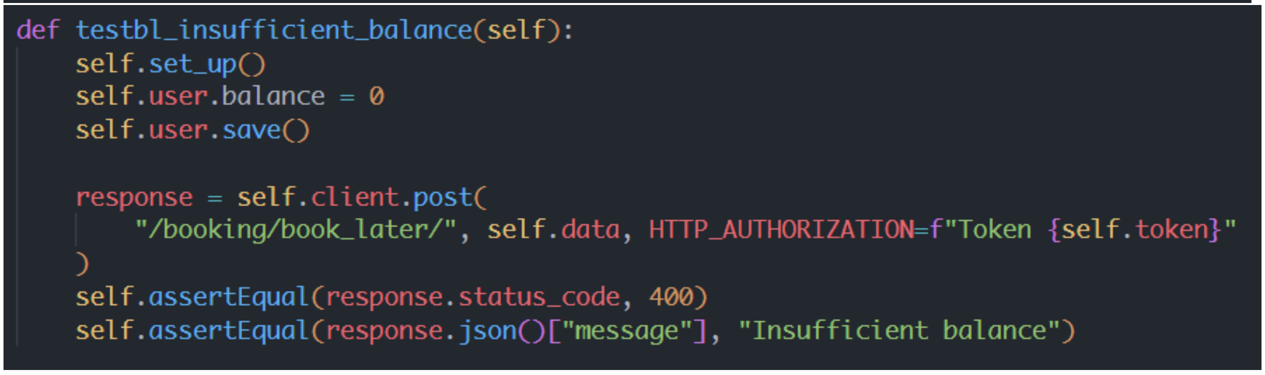


```python
def set_up(self):
    profile_manager = ProfileManager()
    profile_manager.model = Profile

    self.user = profile_manager.create_user(
        email="test@test.com",
        password="testpassword",
        first_name="test",
        last_name="user",
        phone="1234567890",
    )

    self.user.is_active = True
    self.user.is_subscribed = True
    self.user.balance = 2000
    self.user.save()

    data = {
        "email": "test@test.com",
        "password": "testpassword",
    }

    self.hub = Hub.objects.create(
        hub_name="Test Hub", max_capacity=10, latitude=0, longitude=0
    )
    self.cycle = Cycle.objects.create(
        hub=self.hub, user=self.user, booked=False, active=False
    )
    self.lock = Lock.objects.create(
        arduino_port="test", hub=self.hub, cycle=self.cycle
    )

    response = self.client.post("/auth/get_auth_token/", data)
    self.token = response.json()["token"]

    self.new_time = datetime.now() + timedelta(minutes=75)
```

The edge cases are tested as follows

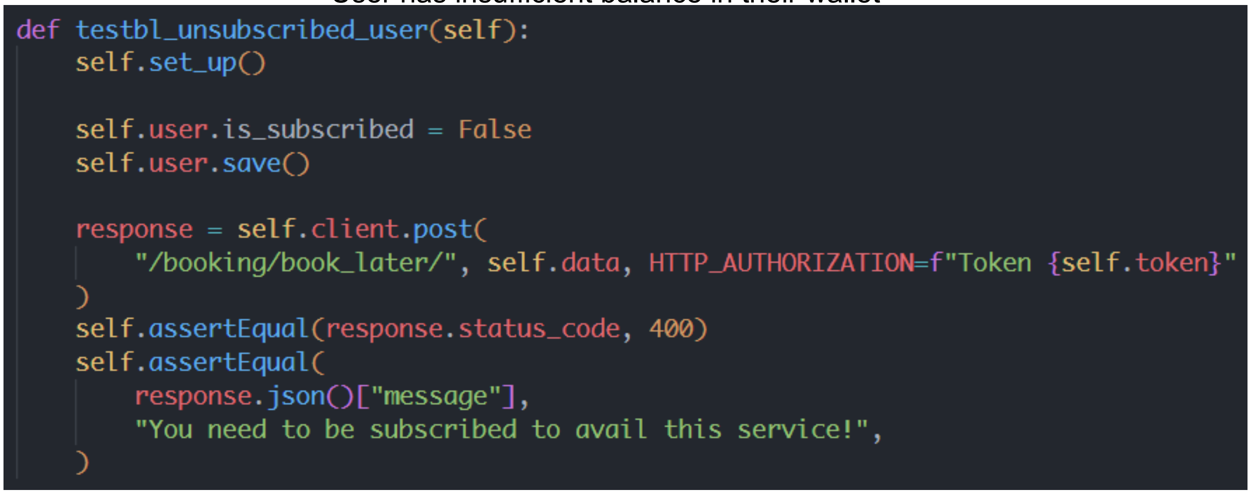


```
def testbl_normal(self):
    self.set_up()

    response = self.client.post(
        "/booking/book_later/", self.data, HTTP_AUTHORIZATION=f"Token {self.token}"
    )
    self.assertEqual(response.status_code, 201)
```



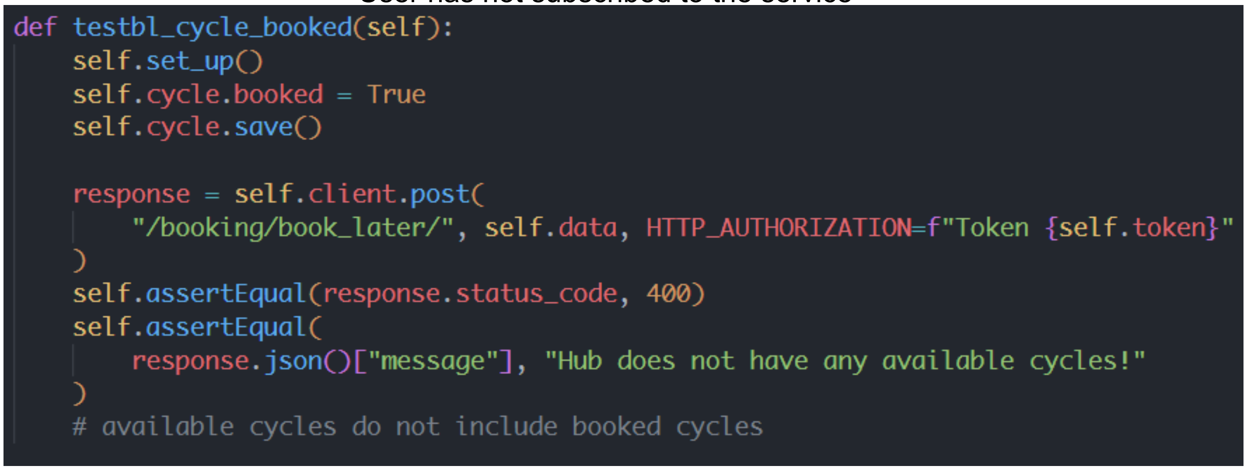
```
def testbl_insufficient_balance(self):
    self.set_up()
    self.user.balance = 0
    self.user.save()

    response = self.client.post(
        "/booking/book_later/", self.data, HTTP_AUTHORIZATION=f"Token {self.token}"
    )
    self.assertEqual(response.status_code, 400)
    self.assertEqual(response.json()["message"], "Insufficient balance")
```

User has insufficient balance in their wallet



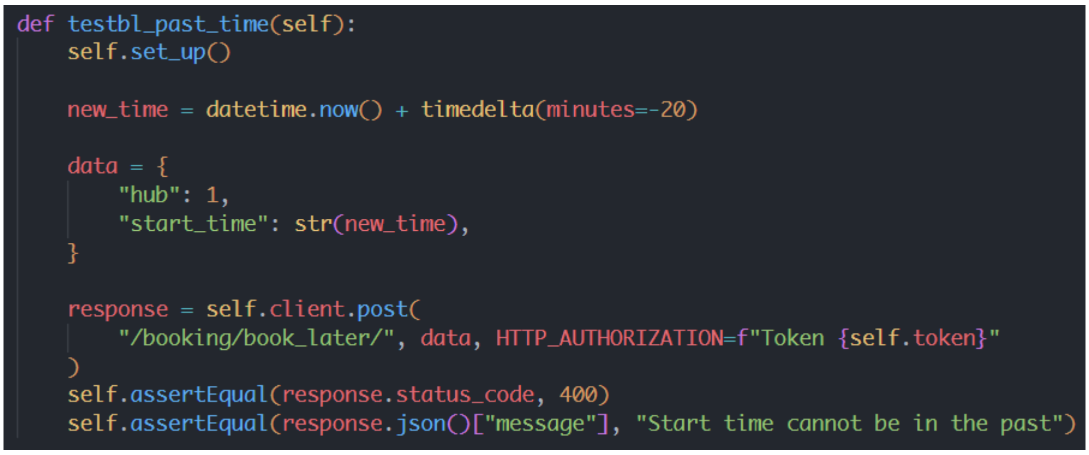
```
def testbl_unsubscribed_user(self):
    self.set_up()

    self.user.is_subscribed = False
    self.user.save()

    response = self.client.post(
        "/booking/book_later/", self.data, HTTP_AUTHORIZATION=f"Token {self.token}"
    )
    self.assertEqual(response.status_code, 400)
    self.assertEqual(
        response.json()["message"],
        "You need to be subscribed to avail this service!",
    )
```

User has not subscribed to the service



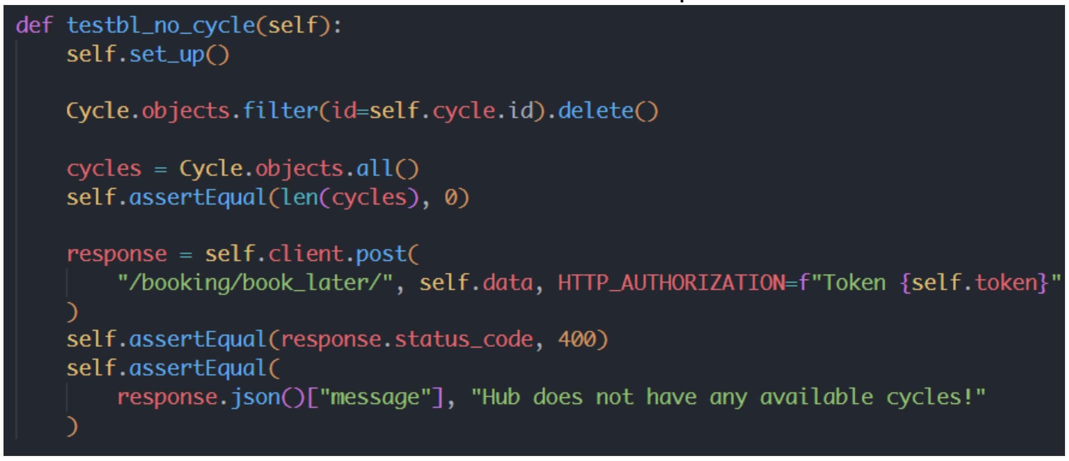
```
def testbl_cycle_booked(self):
    self.set_up()
    self.cycle.booked = True
    self.cycle.save()

    response = self.client.post(
        "/booking/book_later/", self.data, HTTP_AUTHORIZATION=f"Token {self.token}"
    )
    self.assertEqual(response.status_code, 400)
    self.assertEqual(
        response.json()["message"], "Hub does not have any available cycles!"
    )
    # available cycles do not include booked cycles
```

Cycle is already booked by another user



```python
def testbl_past_time(self):
    self.set_up()

    new_time = datetime.now() + timedelta(minutes=-20)

    data = {
        "hub": 1,
        "start_time": str(new_time),
    }

    response = self.client.post(
        "/booking/book_later/", data, HTTP_AUTHORIZATION=f"Token {self.token}"
    )
    self.assertEqual(response.status_code, 400)
    self.assertEqual(response.json()["message"], "Start time cannot be in the past")
```

Desired start time is in the past

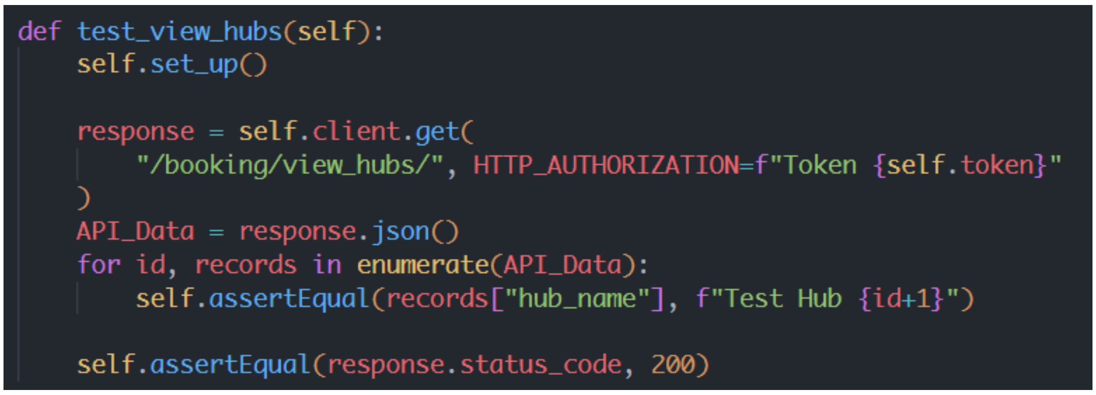


```python
def testbl_no_cycle(self):
    self.set_up()

    Cycle.objects.filter(id=self.cycle.id).delete()

    cycles = Cycle.objects.all()
    self.assertEqual(len(cycles), 0)

    response = self.client.post(
        "/booking/book_later/", self.data, HTTP_AUTHORIZATION=f"Token {self.token}"
    )
    self.assertEqual(response.status_code, 400)
    self.assertEqual(
        response.json()["message"], "Hub does not have any available cycles!"
    )
```

No cycle available at the desired hub

3.2.4 Getting Details of Each Hub

API Endpoint: `/booking/view_hubs/`
Test Owner: Nevish Pathe
Date: 26/03/2024
Test Description: This test case is used to check if a user can get the details of each hub.
Test Results: User is able to get the details of each hub successfully.
The database is pre-populated with various hub details in the `setUp()` method.

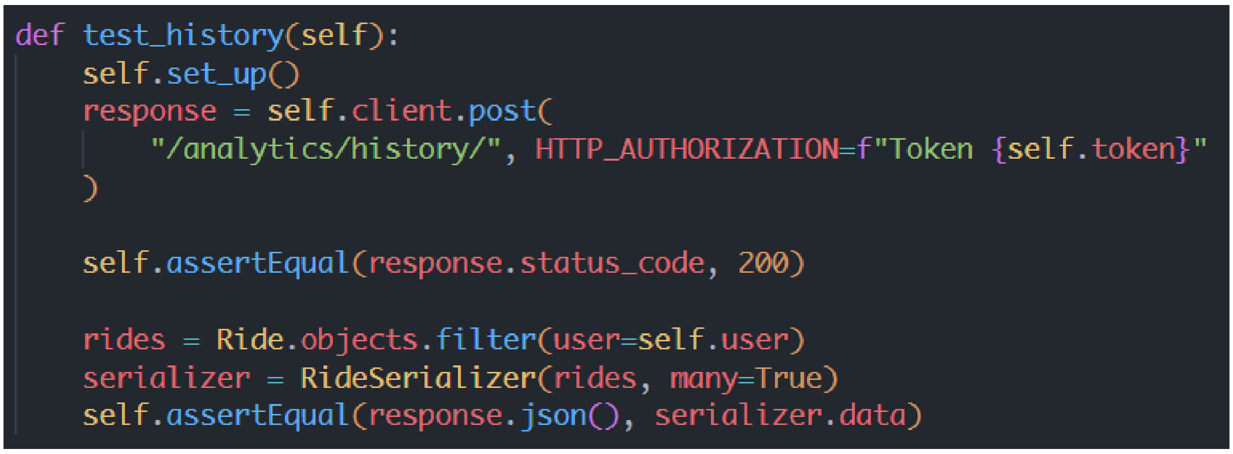


```python
def test_view_hubs(self):
    self.set_up()

    response = self.client.get(
        "/booking/view_hubs/", HTTP_AUTHORIZATION=f"Token {self.token}"
    )
    API_Data = response.json()
    for id, records in enumerate(API_Data):
        self.assertEqual(records["hub_name"], f"Test Hub {id+1}")

    self.assertEqual(response.status_code, 200)
```

3.3 User Analytics

3.3.1 Viewing Ride History

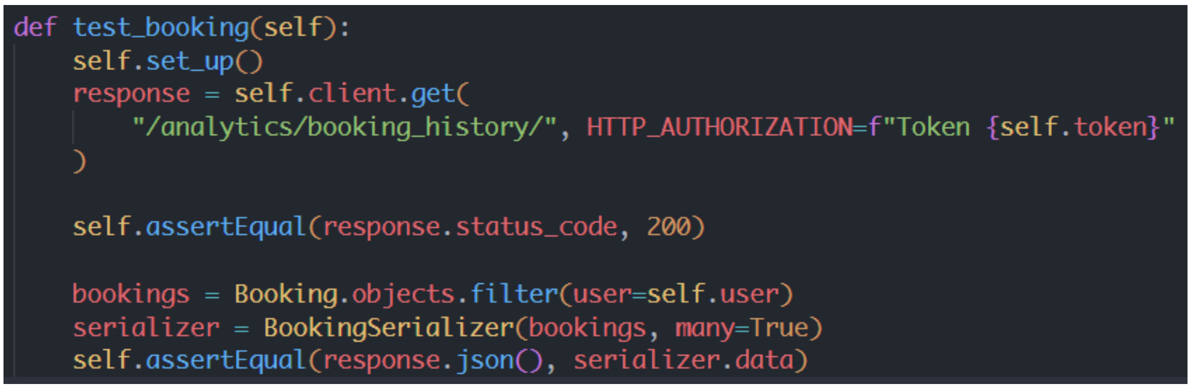
API Endpoint: `/analytics/history/`
Test Owner: Kaneez Fatima
Date: 25/03/2024
Test Description: This test case is used to check if a user can view their ride history.
Test Results: User is able to view their ride history successfully.
The database is initialized with multiple rides in the `setUp()` method.



```python
def test_history(self):
    self.set_up()
    response = self.client.post(
        "/analytics/history/", HTTP_AUTHORIZATION=f"Token {self.token}"
    )

    self.assertEqual(response.status_code, 200)

    rides = Ride.objects.filter(user=self.user)
    serializer = RideSerializer(rides, many=True)
    self.assertEqual(response.json(), serializer.data)
```

3.3.2 Viewing Past Bookings

API Endpoint: `/analytics/booking_history/`
Test Owner: Kaneez Fatima
Date: 25/03/2024
Test Description: This test case is used to check if a user can view their past and active bookings.
Test Results: User is able to view their past bookings successfully.
The database is initialized with multiple bookings in the `setUp()` method.

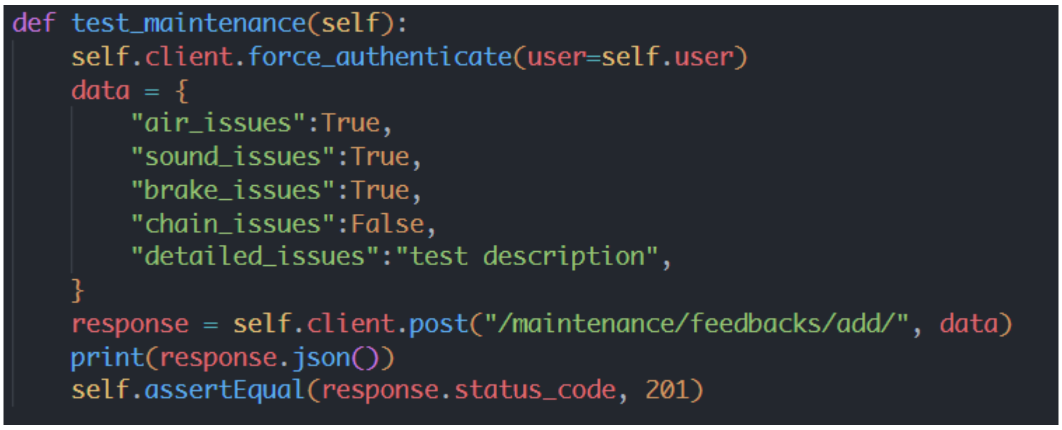


```python
def test_booking(self):
    self.set_up()
    response = self.client.get(
        "/analytics/booking_history/", HTTP_AUTHORIZATION=f"Token {self.token}"
    )

    self.assertEqual(response.status_code, 200)

    bookings = Booking.objects.filter(user=self.user)
    serializer = BookingSerializer(bookings, many=True)
    self.assertEqual(response.json(), serializer.data)
```

3.4 Maintenance

3.4.1 Reporting an Issue

API Endpoint: `/maintenance/feedbacks/add/`
Test Owner: Ananya Baghel
Date: 28/03/2024
Test Description: This test case is used to check if a user can report an issue with the cycle.
Test Results: User is able to report an issue with the cycle successfully.

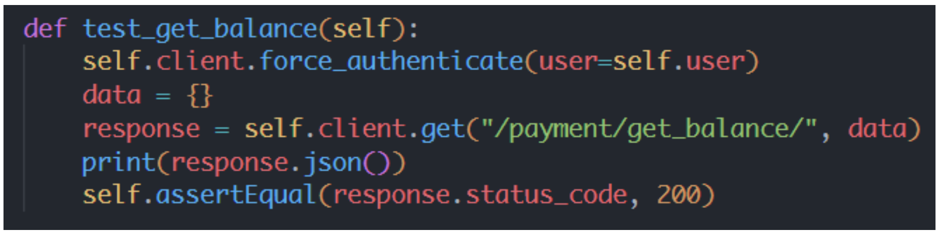


```python
def test_maintenance(self):
    self.client.force_authenticate(user=self.user)
    data = {
        "air_issues":True,
        "sound_issues":True,
        "brake_issues":True,
        "chain_issues":False,
        "detailed_issues":"test description",
    }
    response = self.client.post("/maintenance/feedbacks/add/", data)
    print(response.json())
    self.assertEqual(response.status_code, 201)
```

3.5 Payment

3.5.1 Getting Wallet Balance

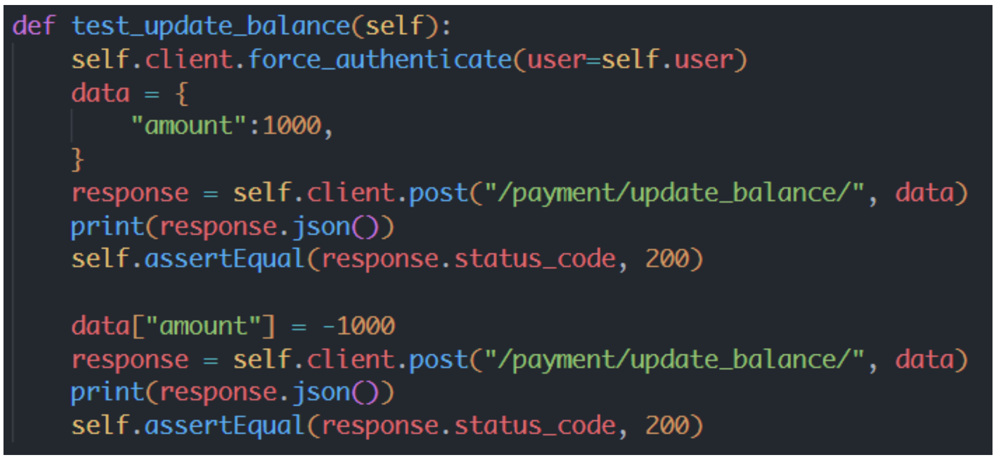
API Endpoint: `/payment/get_balance/`
Test Owner: Ananya Baghel
Date: 28/03/2024
Test Description: This test case is used to check if a user can get their wallet balance.
Test Results: User is able to get their wallet balance successfully.



```python
def test_get_balance(self):
    self.client.force_authenticate(user=self.user)
    data = {}
    response = self.client.get("/payment/get_balance/", data)
    print(response.json())
    self.assertEqual(response.status_code, 200)
```

3.5.2 Updating Wallet Balance

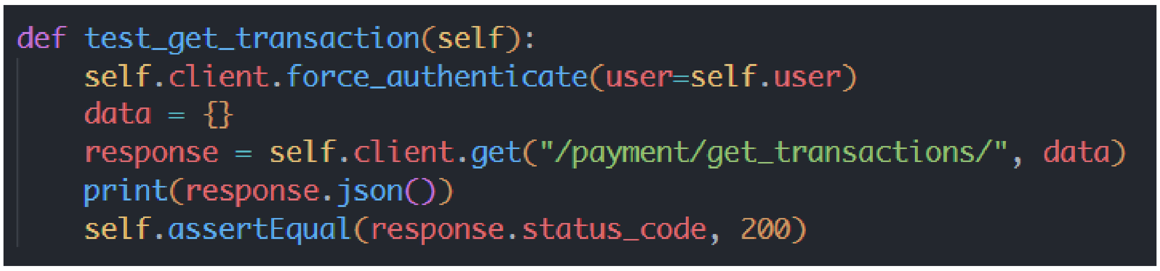
API Endpoint: `/payment/update_balance/`
Test Owner: Ananya Baghel
Date: 28/03/2024
Test Description: This test case is used to check if a user can update their wallet balance.
Test Results: User is able to update their wallet balance successfully.



```python
def test_update_balance(self):
    self.client.force_authenticate(user=self.user)
    data = {
        "amount":1000,
    }
    response = self.client.post("/payment/update_balance/", data)
    print(response.json())
    self.assertEqual(response.status_code, 200)

    data["amount"] = -1000
    response = self.client.post("/payment/update_balance/", data)
    print(response.json())
    self.assertEqual(response.status_code, 200)
```

3.5.3 Viewing Transaction History

API Endpoint: `/payment/get_transactions/`
Test Owner: Ananya Baghel
Date: 28/03/2024
Test Description: This test case is used to check if a user can view their transaction history.
Test Results: User is able to view their transaction history successfully.



```python
def test_get_transaction(self):
    self.client.force_authenticate(user=self.user)
    data = {}
    response = self.client.get("/payment/get_transactions/", data)
    print(response.json())
    self.assertEqual(response.status_code, 200)
```

4. Integration Testing

4.1 Authentication

4.1.1 Registering a User

Module Details: This module integrates the registration process for new users, providing the necessary functionality to create an account.
Test Owner: Srishti Chandra
Test Date: 27/03/2024
Test Results:

- **Test Case 1:** Invalid name, phone number, or email address is entered.
  Result: Error messages are displayed in red text below the respective entry fields:
  - ”Please enter your full name” is displayed if the full name field is left blank or contains only whitespace characters.
  - ”Enter a valid phone number” is displayed if an invalid phone number format is entered.
  - ”Enter a valid email address” is displayed if an invalid email address format is entered.
- **Test Case 2:** Valid name, phone number, and already registered email address are entered in the respective fields.
  Result: An alert dialog is triggered, notifying the user of an existing profile associated with the provided email address.
- **Test Case 3:** Valid name, phone number, and unregistered email address are entered in the respective fields.
  Result: Upon successful registration, the user is directed to the Account Created page. Here, instructions are given to activate the account via an email link. Clicking ”Continue” on this page leads the user to the login page.

Additional Comments: Any trailing whitespace in the Full Name field should be removed and the full name must contain first and last name to ensure that the textbox recognizes the entry as complete.

4.1.2 Logging in a User

Module Details: This module is integration of all the components of the login page.
Test Owner: Srishti Chandra
Test Date: 27/03/2024
Test Results:

- **Test Case 1:** If the user has not signed up and attempts to log in with an unregistered email ID and password.
  Result: An alert prompt is displayed that conveys that no such user exists.
- **Test Case 2:** If the user has not signed up and attempts to sign up.
  Result: The user can click on the ”Not on Pedal Pal Yet? Sign Up!” link located at the bottom of the screen, which redirects them to the registration page.
- **Test Case 3:** Invalid email ID is entered, such as incorrect formatting or other issues.
  Result: An error message in red text is displayed below the email input field, indicating ”Invalid email address format.”

- **Test Case 4:** Incorrect password is entered with a registered email ID.
  Result: An alert prompt is displayed that conveys “Incorrect Password”.
- **Test Case 5:** User doesn’t remember the password.
  Result: Clicking on ”Forgot Password” redirects to the Forgot Password page.
- **Test Case 6:** Registered email ID and password are entered, and the Login button is clicked.
  Result: The user is successfully redirected to the Homepage of Pedalpal.

Additional Comments: None

4.1.3 Email Verification

4.1.4 Forgot Password

4.2 Wallet: Transaction History and Add Balance

Module Details: This module includes tests for:

- Transaction Logs of the in-app Wallet, featuring payments made while using app.
- Functionality of adding balance to the wallet.

Test Owner: Raghav Manglik
Test Date: [26-03-2024 - 27-03-2024]
Test Results:

- **Test case 1:** An unsubscribed user attempts to access the wallet, which is a service that requires a subscription.
  Result: An alert prompt is displayed telling user that this feature is available only for subscribed users.
- **Test case 2:** After subscribing, A user tries to access the wallet.
  Result: My Wallet page is displayed, showing the current balance after deducting subscription fee of Rs.20 from predetermined amount of Rs.100 along with option to Add balance and View Transaction History
- **Test case 3:** A user tries to view Transaction History.
  Result: Transaction history page displaying credit and debit of balance with green and red colour respectively are shown.
- **Test case 4:** A user tries to add Rs.0 to wallet.
  Result: The Razorpay payment gateway fails to operate correctly without displaying any error messages. (DEBUG NEEDED)
- **Test case 5:** A user tries to copy/cut and paste an invalid amount (-ve amount or a string of characters) in Add Balance window.
  Result: Only leading numerical values are pasted in the Enter Amount box.
- **Test case 6:** A user tries to enter more than 2 digits after the decimal point / any symbol other than a single decimal point in Enter Amount Box in Add Balance window.
  Result: The input restricts user to enter no more than two digits after the decimal point, while also restricting the entry of invalid symbols and permitting only a single decimal point.

Additional Comments: [WHAT TO BE CHANGED AFTER DEBUG]

4.3 Ride History Page

Module Details: This module includes tests for viewing bicycle renting logs.
Test Owner: Raghav Manglik
Test Date: [26-03-2024 - 27-03-2024]
Test Results:

- **Test Case 1:** A user who hasn't rented bicycles before accesses the History page.
  Result: History Page is displayed with Total time used 0h 0m and no record of any ride.
- **Test Case 2:** A user while riding, that is with an active ride, accesses the History Page.
  Result: The History Page is displayed with information regarding past rides and no details about the active ride.

Additional Comments: The discrepancy between the starting and ending times compared to the duration shown in the middle below the arrow arises from the system factoring in seconds in its calculation.

4.4 My Booking Page

Module Details: This module includes tests for accessing logs for both past bicycle bookings and currently active bookings.
Test Owner: Raghav Manglik
Test Date: [26-03-2024 - 27-03-2024]
Test Results:

- **Test Case 1:** An unsubscribed user accesses My Bookings page, a subscription required service.
  Result: My Bookings Page is displayed with both Current and Past Bookings as 0.
- **Test Case 2:** A user who hasn't booked any bicycle before accesses My Bookings page.
  Result: My Bookings Page is displayed with both Current and Past Bookings as 0.
- **Test Case 3:** A Bicycle is booked for a finite time and My Bookings Page is accessed.
  Result: My bookings page is displayed with current bookings incremented and showing the details about the booking. Initially, The bicycle remained in Current Bookings section once the booking period expired. This was subsequentially rectified as once the booking period expired, The bicycle moved to Past Bookings section.

Additional Comments: The discrepancy between the starting and ending times of the booking compared to the duration shown arises from the system factoring in seconds in its calculation.

4.5 View Profile Page and Subscribe

Module Details: This module includes tests for accessing information about a user.
Test Owner: Raghav Manglik
Test Date: [26-03-2024 - 27-03-2024]
Test Results:

- **Test Case 1:** A user navigates to their Profile Page.
  Result: Profile Page is displayed with their Name, Email, Phone, Subscription Status and a default Profile picture. If the user is not subscribed, an additional button is displayed on their Profile Page to navigate to the subscription option. Clicking this button opens the Razorpay payment gateway, which charges Rs.100. Out of this amount, Rs.80 is added to the user's wallet, and Rs.20 is deducted as the subscription fee.

- **Test Case 2:** A user attempts to change their profile picture by clicking on a camera icon beside the profile picture.
  Result: The feature to change Personal Information along with Profile Picture is yet to be implemented. Currently clicking on that icon does nothing.

Additional Comments:

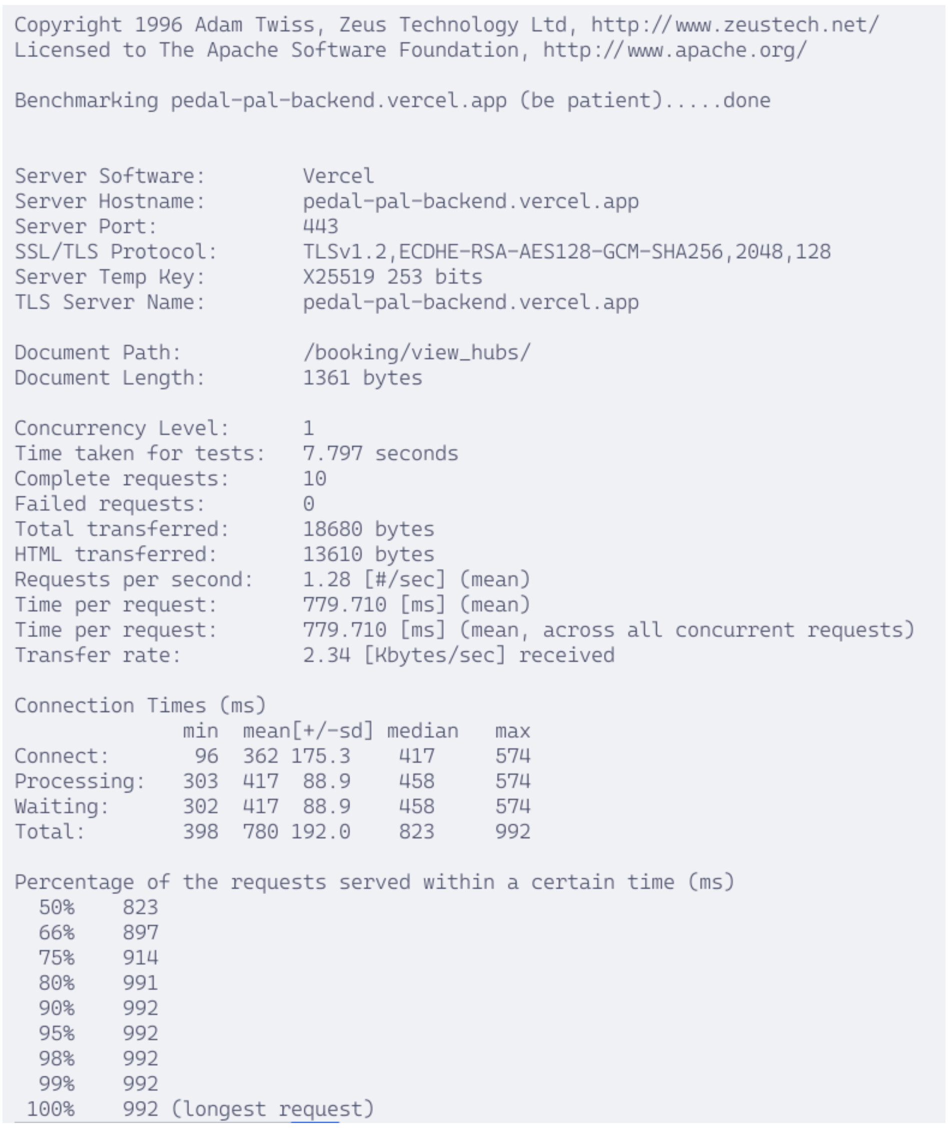
5. System Testing

5.1 Functional Requirements

5.2 Non-Functional Requirements

5.2.1 Performance Requirements

Test Owner: Amogh Bhagwat
Date: 30/03/2024
Test Description: This test case is used to check the response time of the backend server.
Test Results: The response time of the backend server APIs is less than 1 second.



```
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking pedal-pal-backend.vercel.app (be patient).....done


Server Software:        Vercel
Server Hostname:        pedal-pal-backend.vercel.app
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES128-GCM-SHA256,2048,128
Server Temp Key:        X25519 253 bits
TLS Server Name:        pedal-pal-backend.vercel.app

Document Path:          /booking/view_hubs/
Document Length:        1361 bytes

Concurrency Level:      1
Time taken for tests:   7.797 seconds
Complete requests:      10
Failed requests:        0
Total transferred:      18680 bytes
HTML transferred:       13610 bytes
Requests per second:    1.28 [#/sec] (mean)
Time per request:       779.710 [ms] (mean)
Time per request:       779.710 [ms] (mean, across all concurrent requests)
Transfer rate:          2.34 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       96  362 175.3    417     574
Processing:   303  417  88.9    458     574
Waiting:      302  417  88.9    458     574
Total:        398  780 192.0    823     992

Percentage of the requests served within a certain time (ms)
  50%    823
  66%    897
  75%    914
  80%    991
  90%    992
  95%    992
  98%    992
  99%    992
 100%    992 (longest request)
```

Additional Comments: The response time of the backend server APIs is less than 1 second only when the server is already active. Since we are using Vercel's free plan, the server goes to sleep after 5 minutes of inactivity. The first request after the server wakes up takes around 10-30 seconds.

5.2.2 Security Requirements

- All user passwords are encrypted before storing them in the database. The user's password is never stored in plain text.
- All payments are done through Razorpay's secure payment gateway.
- All API calls between frontend and backend are authenticated using JWT tokens, and are encrypted using HTTPS.

6. Conclusion

A. Group Log

S.No	Date	Timings	Venue	Description
1	07/01/2024	14:00 to 16:00	RM Building	Brain-stormed various possible prospective ideas for the project. Main ideas presented were: Bicycle Rental Services Hall Management Used goods Buy/Sell Portal
2	09/01/2024	14:30 to 17:00	Google Meet	Finalized the idea for the project and discussed various aspects of it.
3	11/01/2024	22:00 to 00:00	Google Meet	Studied the SRS template given and distributed the work amongst the team members.
4	17/01/2024	21:00 to 21:30	Google Meet	First meet with the Teaching Assistant Mr. Bharat. Discussed about project and the SRS documentation.
5	20/01/2024	15:00 to 18:00	Google Meet	Brainstorming of final ideas and discussion on use cases, features, data flows.
6	21/01/2024	14:00 to 15:00	Google Meet	Decided to use Django with bootstrap and inline CSS to build the front end part of our application and PostgreSQL as our DBMS.
7	22/01/2024	23:00 to 00:00	Google Meet	Explored more functionalities for the product and progressed with the SRS document.
8	25/01/2024	14:00 to 17:00	RM Building	Finalized the SRS Document and completed typesetting in $\LaTeX$