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: **April 1, 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 Software Test Document	01/04/2024

2. Introduction

2.1 Test Strategy

We utilized Django's test framework specifically for unit testing, where we examined individual components of our software independently. This automated process ensured that each component functioned correctly on its own.

The integration testing was done manually. We carefully checked all parts of the application, making sure everything worked well together and addressing potential user problems upfront.

2.2 Testing Timeline

We mainly tested our software after finishing the implementation phase, especially using Django's testing framework for automation. However, we also did some manual testing while working on the implementation.

2.3 Testers

The developers themselves conducted the testing process. However, we made sure that the individual responsible for implementing a specific functionality did not conduct the testing for that functionality.

2.4 Coverage Criteria

We used branch coverage for unit testing.

2.5 Tools used for testing

We utilized APItestcase, a Django library, for testing purposes. This allowed us to effectively test the API functionalities of our software. We also employed ApacheBench (ab) for load testing our web servers, utilizing it to simulate heavy traffic and gauge server performance accurately.

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.

```python
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

```
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

```python
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)
```

```python
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

```python
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

```python
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.

```
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.

```
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.

```
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.

```
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. Also in some Android, the app itself successfully verifies the email id of the newly registered user.

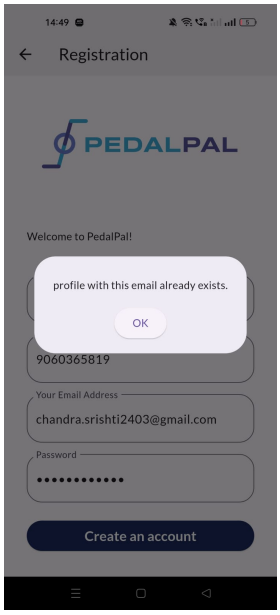
Additional Comments: Any trailing whitespace in the Full Name field should be removed and the full name must contain first and last name to ensure that the textbox recognizes the entry as complete.

4.1.2 Logging in a User

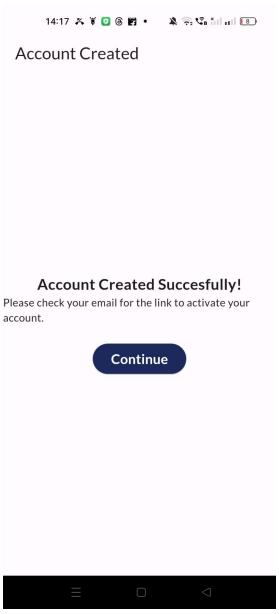
Module Details: This module is integration of all the components of the login page.
Test Owner: Srishti Chandra
Test Date: 27/03/2024
Test Results:

- **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.

(a) Invalid Details Entered

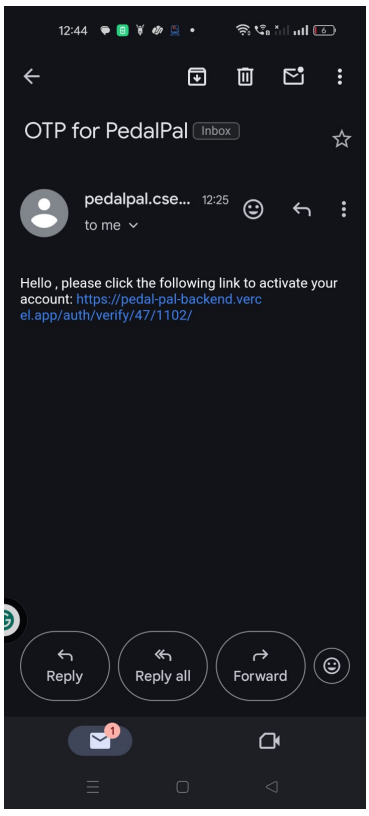


(b) Registered email id entered

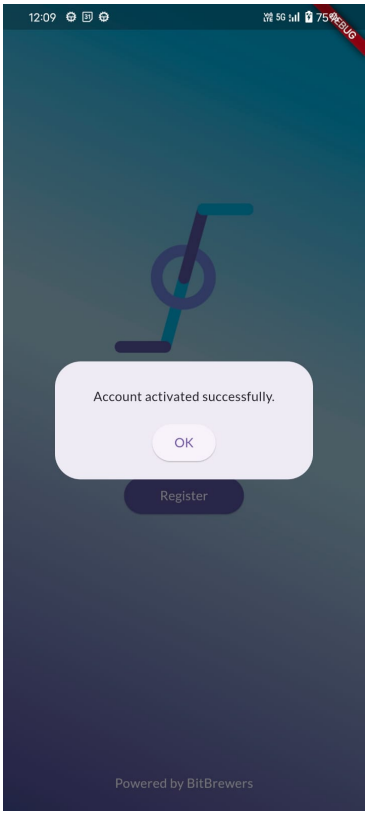


(c) Account Created

Figure 1: Registration



(a) Email verification mail



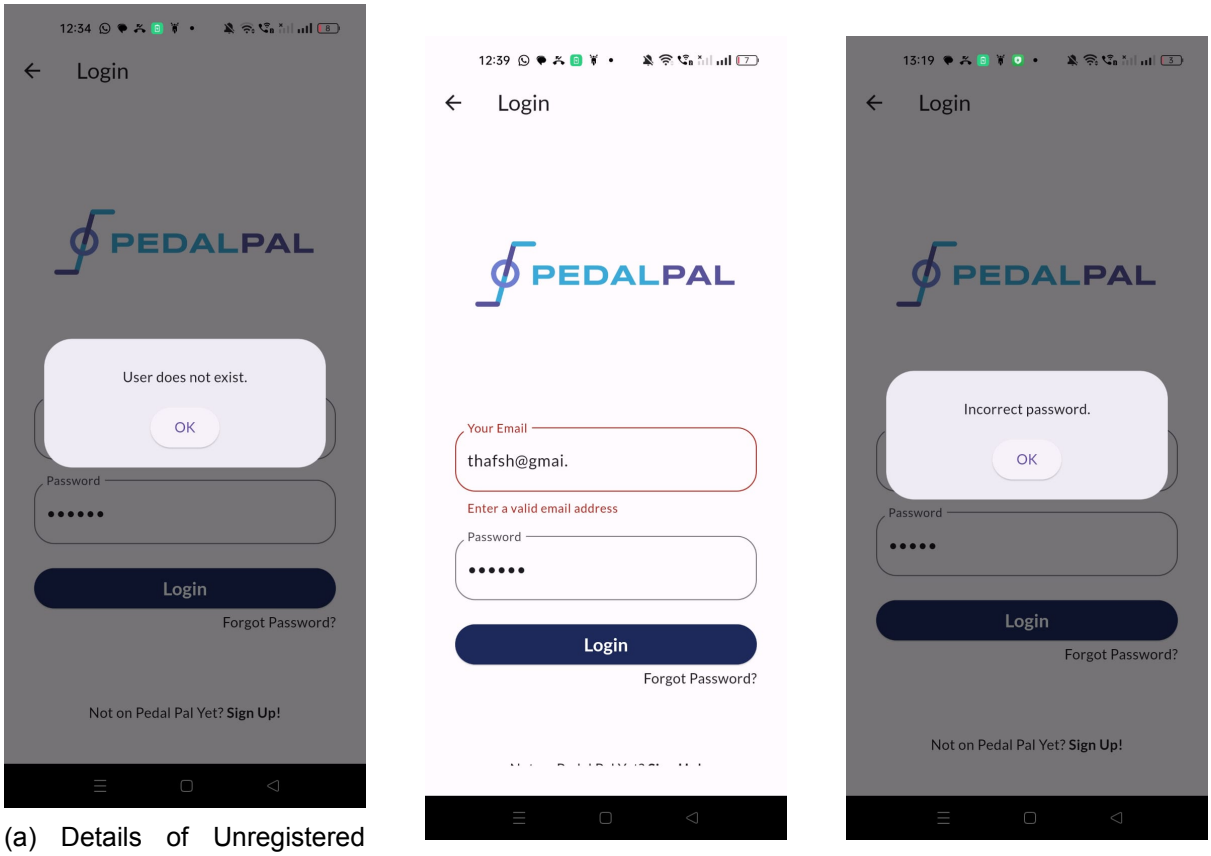
(b) Email Id verified by the app

Figure 2: Registration

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



(a) Details of Unregistered User Entered

(b) Invalid Email Address

(c) Incorrect Password Entered

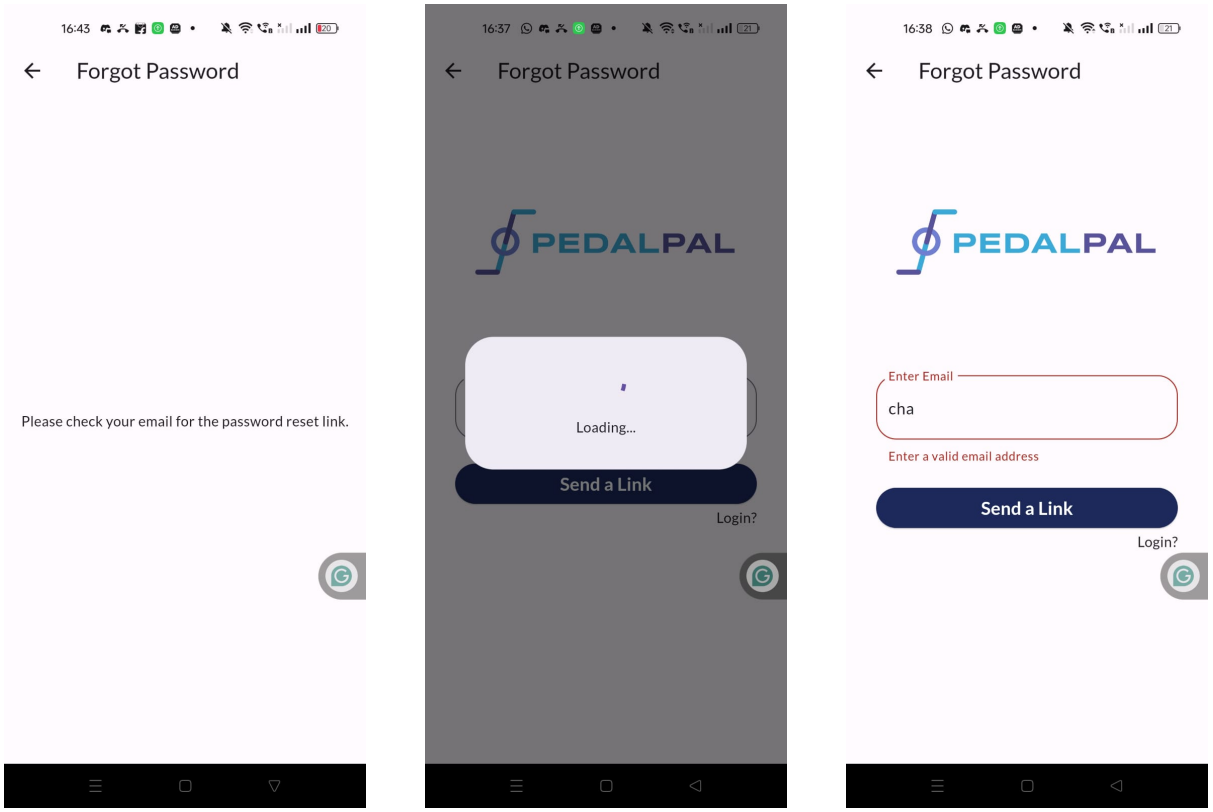
Figure 3: Logging In

4.1.3 Forgot Password

Module Details: This module integrates the forgot password page and all the components that are necessary for the user to update their password.
Test Owner: Srishti Chandra
Test Date: 27/03/2024
Test Results:

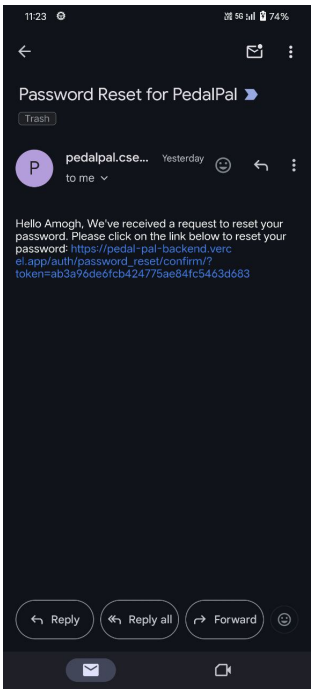
- **Test Case 1:** User initiates a password reset by providing a valid email address and clicking the ”Send a Link” button.
  Result: Upon submission, the user is redirected to a new page instructing them to check their email for the password reset link. Simultaneously, an email containing a password reset link is dispatched to the provided email address. Clicking on the link within the email directs the user to a password reset page, facilitating the entry of a new password along with the token received earlier. Upon submission, the password is reset.
- **Test Case 2:** User enters an invalid email address format (e.g., missing ”@” symbol) and clicks ”Send a Link” button.
  Result: An error message in red is displayed below the entry field, alerting the user to enter a valid email address.
- **Test Case 3:** An unregistered email ID is entered.
  Result: On clicking ”Send a link” button appropriate error message is shown informing user of unregistered email ID.

Additional Comments: None

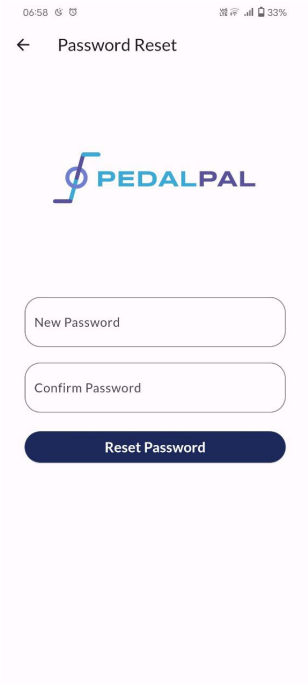


(a) Link Sent to Email (b) Slow Internet Connection (c) Invalid Email Address

Figure 4: Forgot Password



Password Reset Mail



Create New Password

4.2 Ride

4.2.1 Start Ride

Module Details: This module integrates all the components necessary for staring a ride.
Test Owner: Srishti Chandra
Test Date: 27/03/2024
Test Results:

- **Test Case 1:** Upon initiation of the ”Ride Now” functionality from the home screen of the application, the camera interface is invoked,
  Result: Camera opens up, allowing PedalPal to scan the QR code on the lock.
- **Test Case 2:** Upon initiation of the ”Ride Now” functionality from the home screen of the application, the camera interface is not invoked.
  Result: The android asks permission from the user to unblock the camera device so that the QR code can be scanned.
- **Test Case 3:** Initiation of the ”Ride Now” feature while the user possesses an ongoing ride

  Result: A message appears notifying that there is already an active ride for the user . Hence preventing the user to issue multiple cycles at once.
- **Test Case 4:** The QR code on an already rented cycle lock is scanned

  Result: The screen displays that there is an error, hence making this operation unsuccessful.
- **Test Case 5:** The QR code on an available cycle lock is scanned

  Result: The ride starts successfully and the user is redirected to the Ride Service page

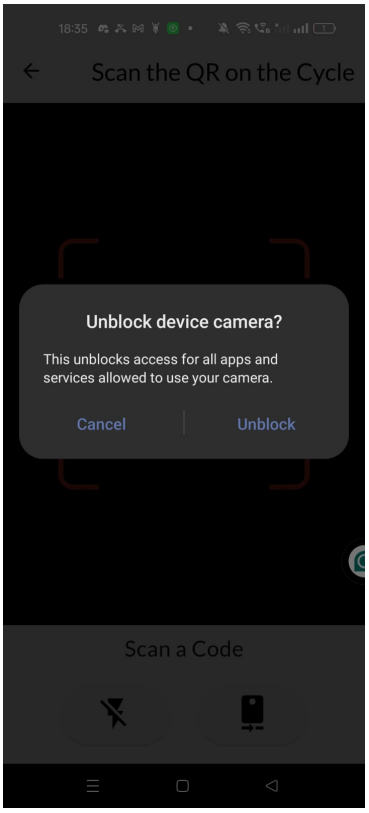
that displays the all information about the currently active ride, serving as the default activation of the ”View Active Ride” feature

- **Test Case 6:** An invalid QR is scanned

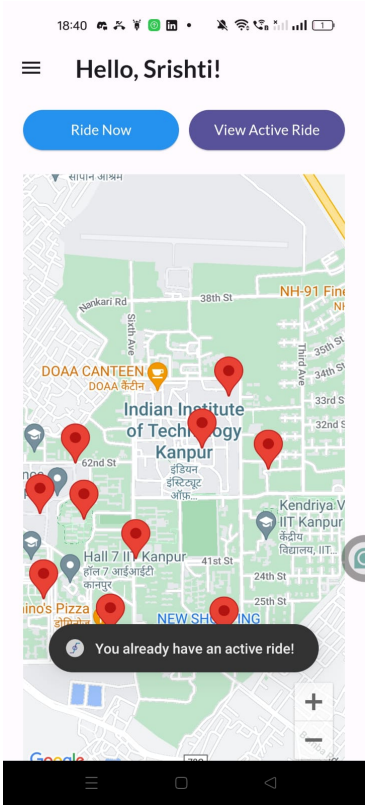
  Result: Error message is prompted by the system indicating the occurrence of an unsuccessful operation.

Additional Comments: None

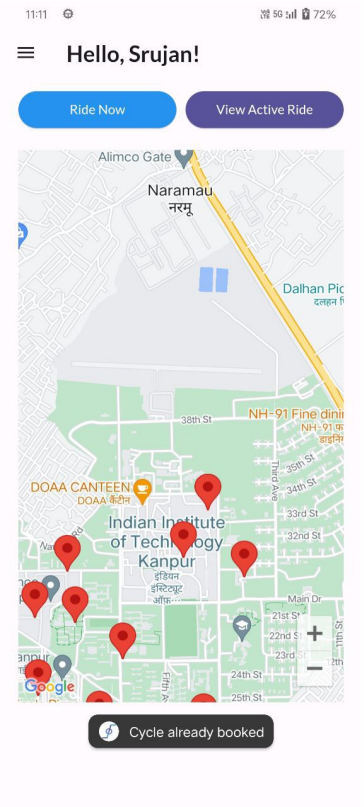
(a) QR scanner opens up



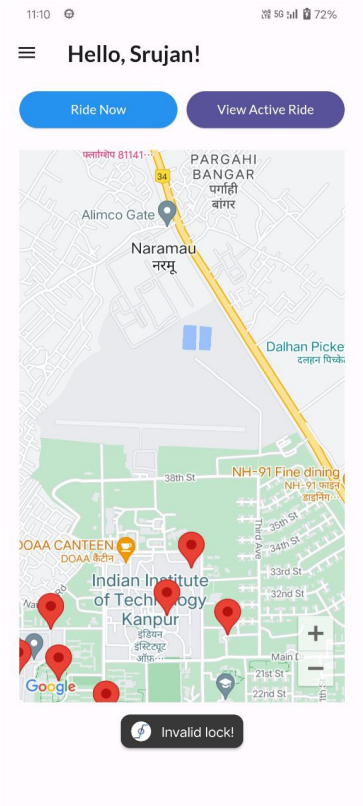
(b) Camera interface blocked



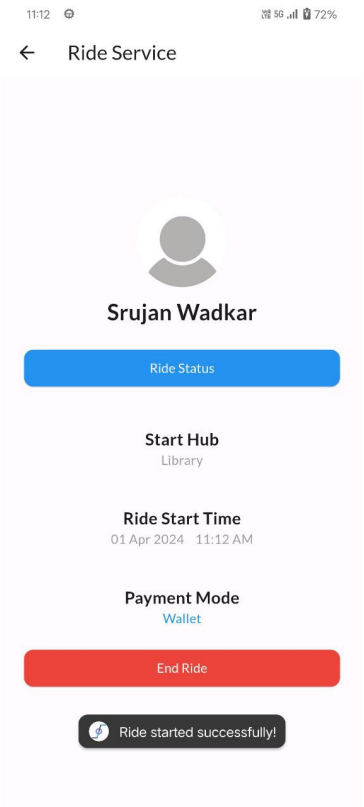
(c) Ride Now is clicked when there is already an active ride



(a) QR of already rented cycle lock scanned



(b) Invalid QR code scanned



(c) Valid QR code of unrented cycle lock scanned

Figure 6: Ride Now

4.2.2 End Ride

Module Details: This module integrates all the components necessary for ending a ride.
Test Owner: Debraj Karmakar
Test Date: 27/03/2024
Test Results:

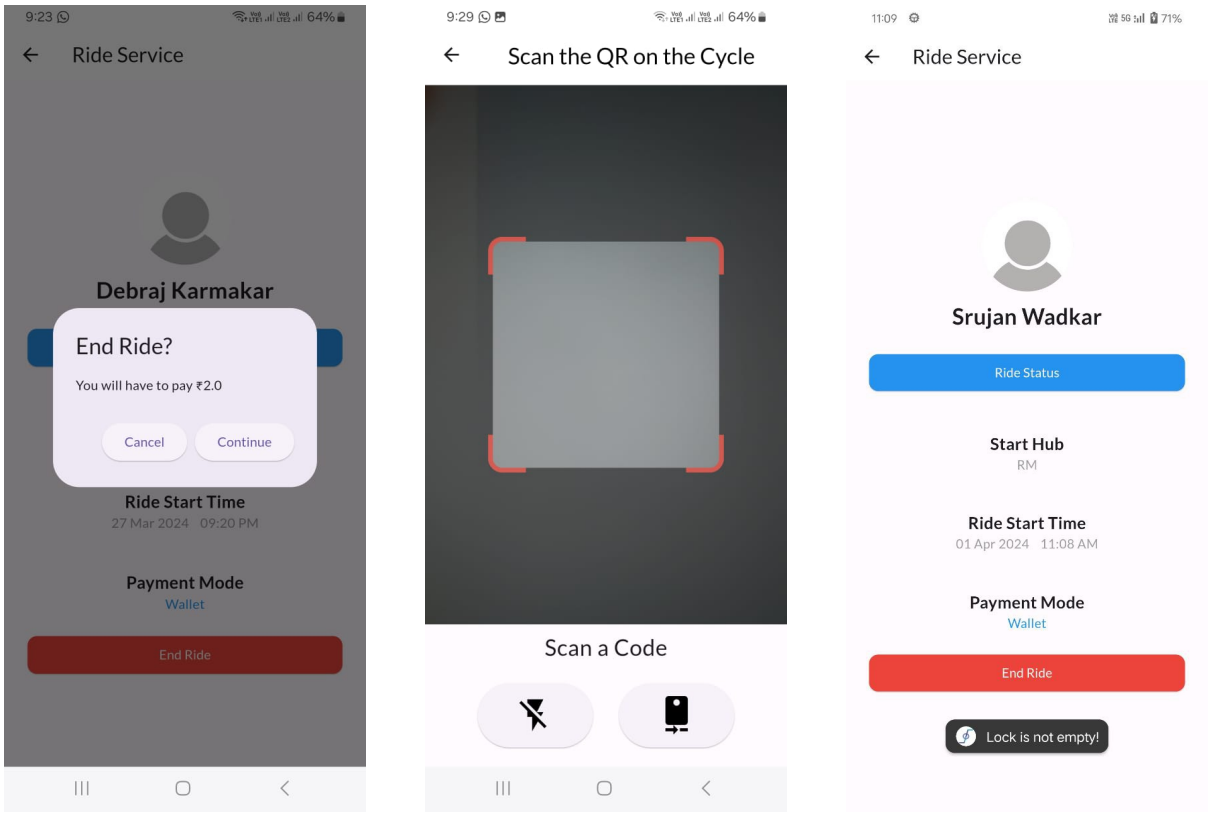
- **Test Case 1:** Upon clicking the “End Ride” button under the ‘View Active Ride’ section of the main screen.
  Result: Gives a prompt for the money to be paid which is calculated by the duration of booking.
- **Test Case 2:** Upon clicking the ‘Cancel’ button.
  Result: The process of end ride is cancelled and the user gets back to the ‘View Active Ride’ section.
- **Test Case 3:** Upon clicking the ‘Continue’ button.
  Result: The user is requested to scan the QR code on lock to which cycle is submitted.
- **Test Case 4:** After scanning the lock, if the lock is not empty.

  Result: System prompts a message that “Lock is not empty!” and redirects to the ‘View Active Ride’ section.

- **Test Case 5:** If lock is empty.
  - **Test Case 5.1:** User is a subscriber.
    Result: System deducts the money from the wallet, ends the ride and asks for feedback.
  - **Test Case 5.2:** Otherwise
    Result: User is redirected to the payment gateway. If payment is successfully completed, the ride ends and the user is asked for feedback. In case payment is not successfully completed, the ride is not ended.

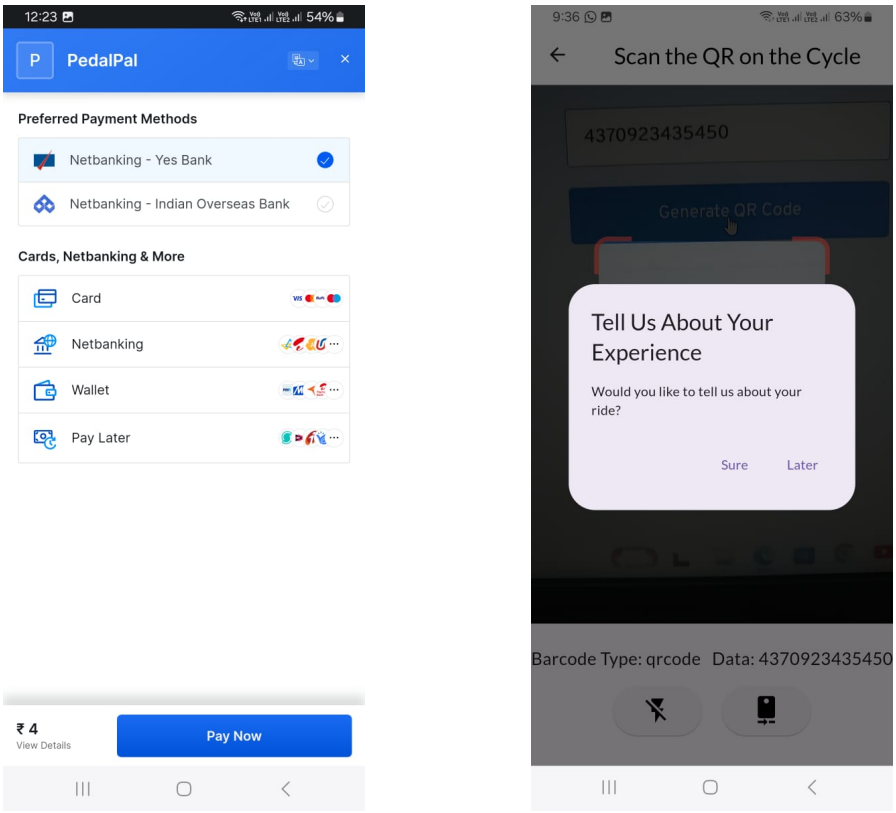
  Users can provide their feedback by clicking on the 'Sure' button or can skip it by clicking on the 'Later' button.

Additional Comments: None



(a) End Ride dialogue box (b) QR code scanner (c) Lock is not empty

Figure 7: End Ride



(a) Payment interface (b) Feedback dialogue box

Figure 8: End Ride

4.3 Wallet: Transaction History and Add Balance

Module Details: This module integrates :

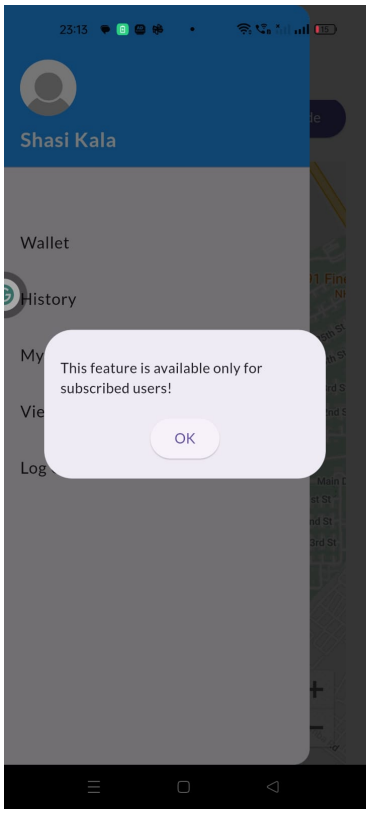
- Transaction Logs of the in-app Wallet, featuring payments made while using app.
- Functionality of adding balance to the wallet.

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

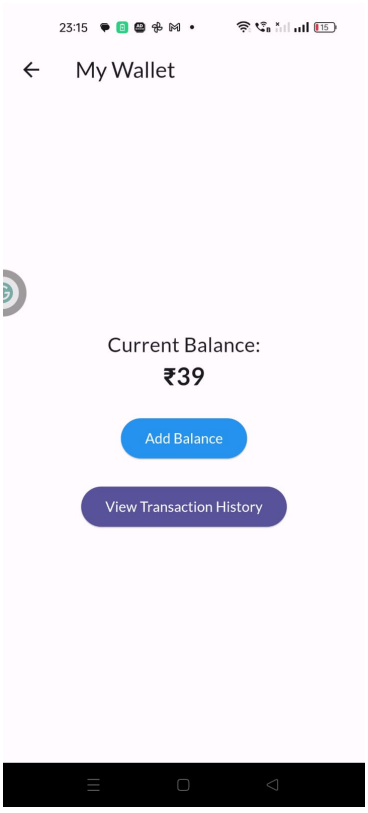
- **Test case 1:** An unsubscribed user attempts to access the wallet, which is a service that requires a subscription.
  Result: An alert prompt is displayed telling user that this feature is available only for subscribed users.
- **Test case 2:** After subscribing, A user tries to access the wallet.
  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 displays an error message asking the user to enter a positive amount.
- **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.

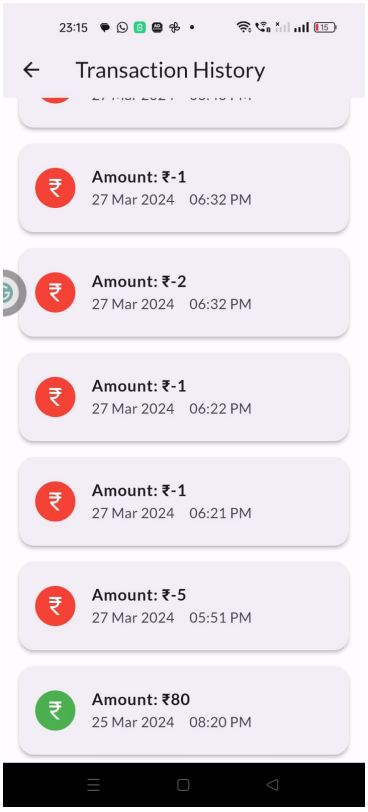
Additional Comments: None



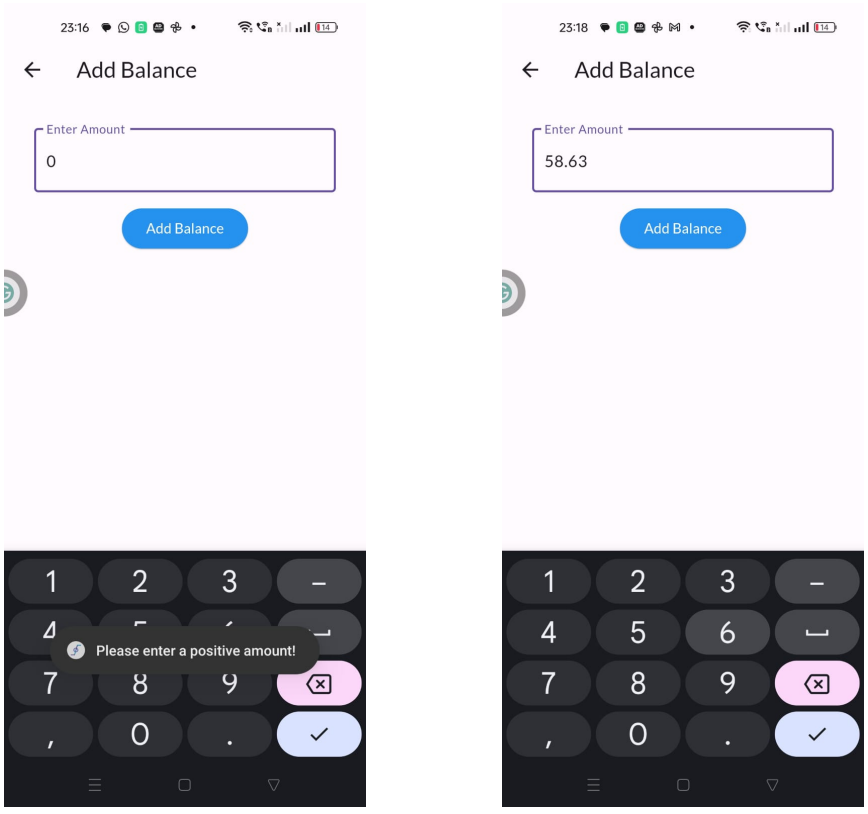
(a) Unsubscribed User Opens Wallet



(b) Wallet Page of a subscriber



(c) Transaction History Page Opens Up



(a) Non-positive amount is entered

(b) Decimal places up to 2 digits are only allowed

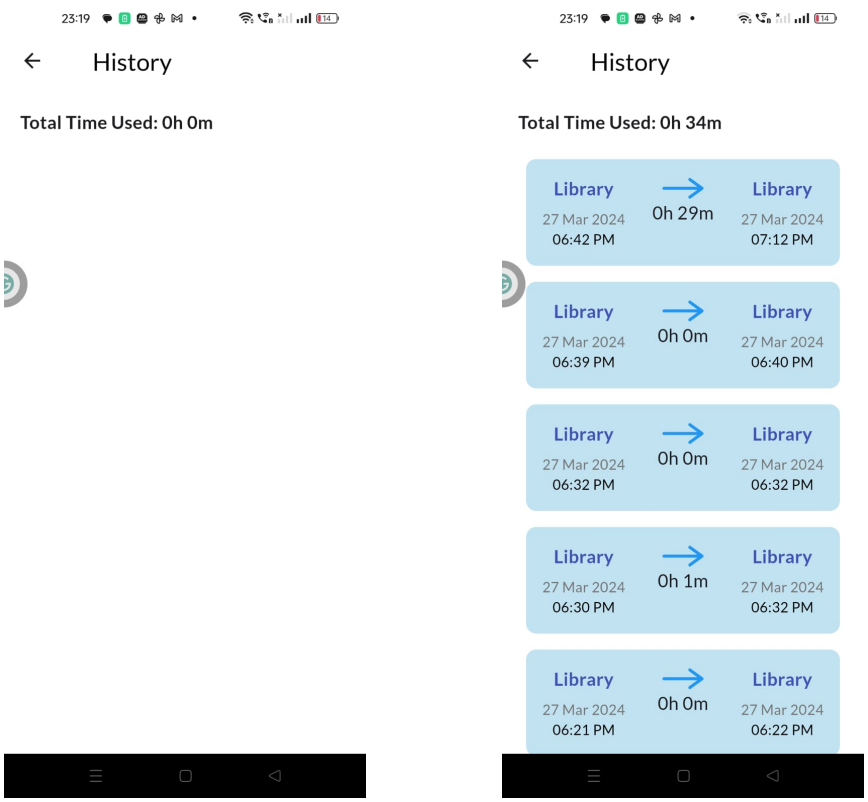
Figure 10: Wallet

4.4 Ride History Page

Module Details: This module integrates all components 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.



(a) No History (b) Non-empty history page

Figure 11: Ride History

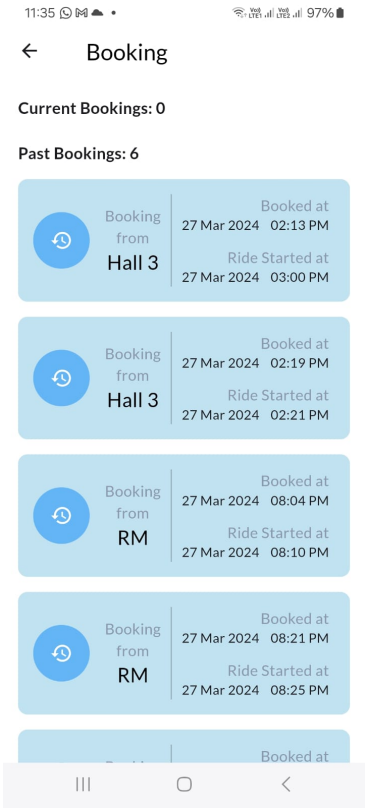
4.5 My Booking Page

Module Details: This module integrates all components necessary 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.

(a) Unsubscribed user/No past bookings



(b) Bookings Page with past bookings

Figure 12: Bookings Page

4.6 View Profile Page and Subscribe

Module Details: This module integrates method 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:

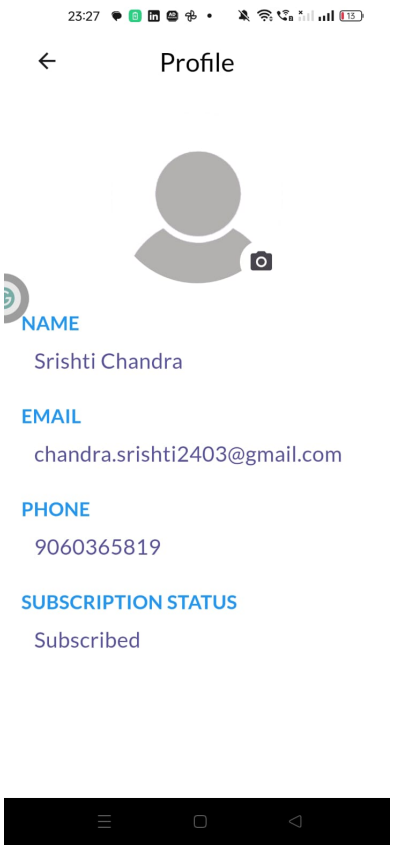


Figure 13: Profile Page Opens Up

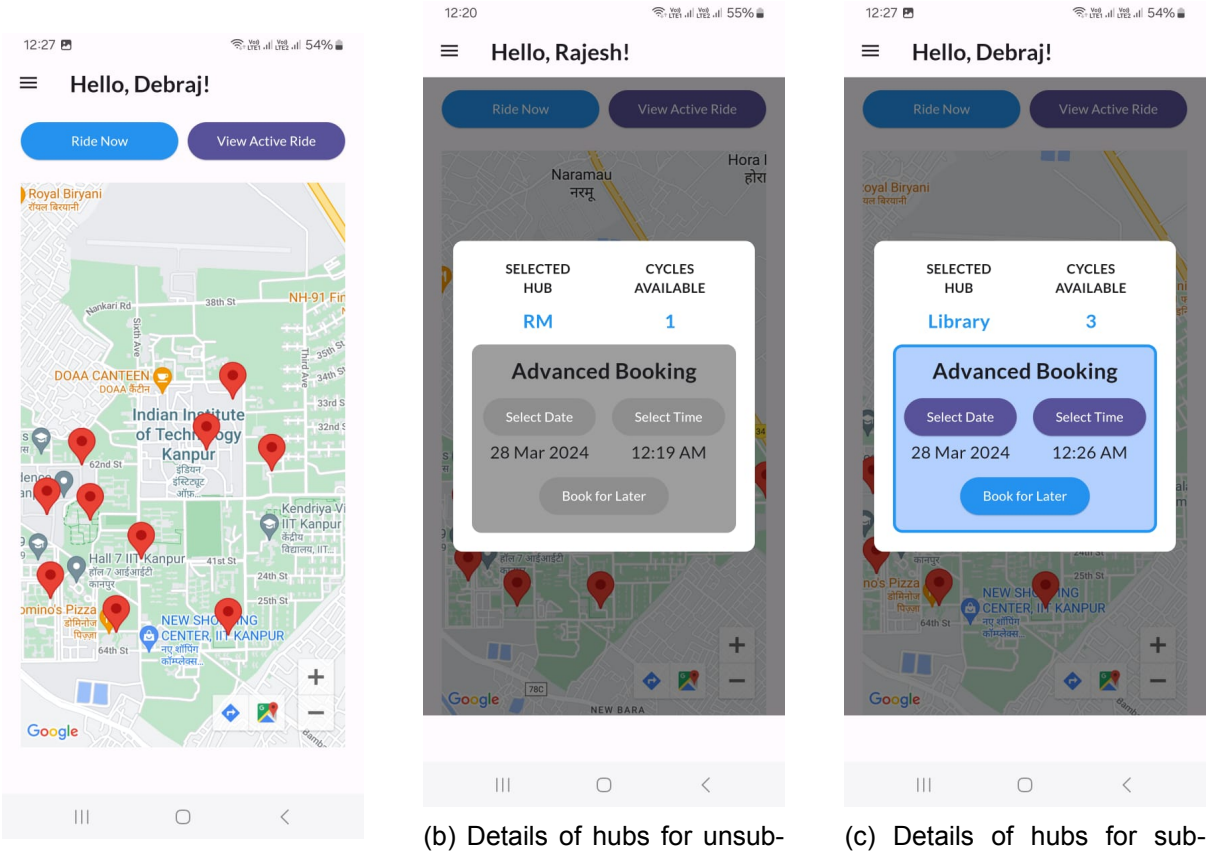
4.7 Viewing Hubs on Map

Module Details: This module integrates the map on the application with real time data of cycle hubs
Test Owner: Debraj Karmakar
Test Date: [26-03-2024 - 27-03-2024]
Test Results:
The available cycle hubs are indicated by red markers on the map.

- **Test Case 1:** Upon clicking on a red marker.
  Result: Displays the details of that hub. Details include name of the hub and number of cycles available on that hub. These details are updated as soon as they change in the database.
- **Test Case 2:** If user is a subscribed user and he/she clicks on the red markers.
  Result: If the user is a subscribed user then along with the above features, the option for “Advance Booking” is available in this dialog box.
- **Test Case 3:** Zooming in and out by +/- buttons.
  Result: The map can be zoomed in or out by clicking on the + (zoom in) and - (zoom out) button on the bottom right corner of map. Other details of the map are adjusted automatically according to the amount of zoom of the map.
- **Test Case 4:** Zooming in and out by finger gestures.

Result: The map can be zoomed in or out by finger gestures by increasing the distance (zoom in) or decreasing the distance (zoom out) between fingers while touching the screen containing the map. Other details of the map are adjusted automatically according to the amount of zoom of the map.

Additional Comments: None

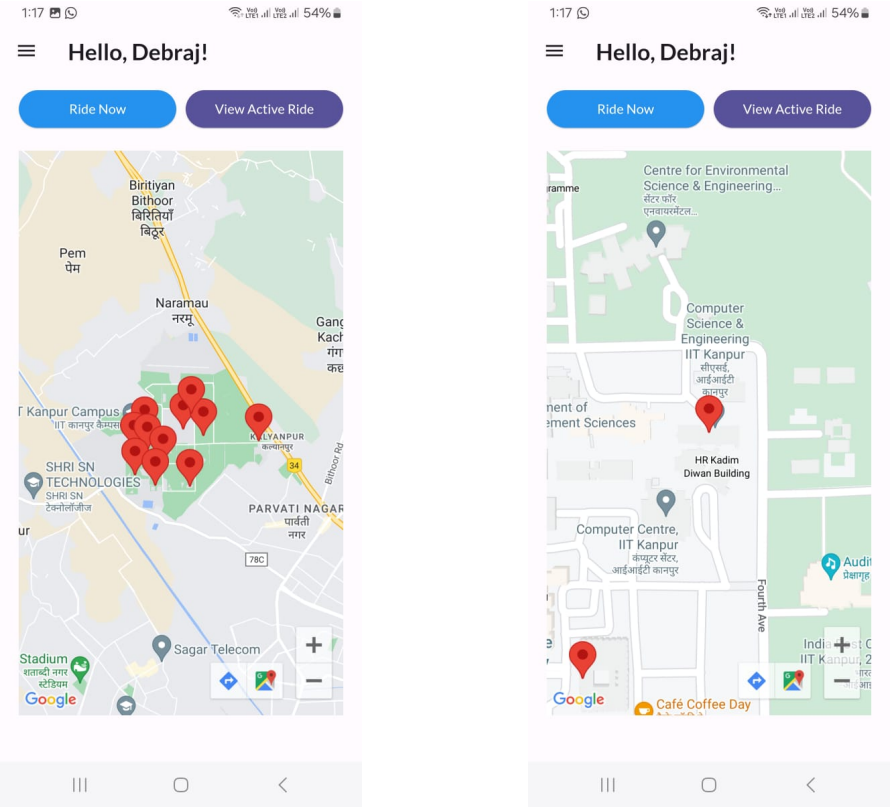


(a) Map Interface

(b) Details of hubs for unsubscribed users

(c) Details of hubs for subscribed users

Figure 14: Viewing Hubs on Map



(a) Zoomed out map (b) Zoomed in map

Figure 15: Viewing Hubs on Map

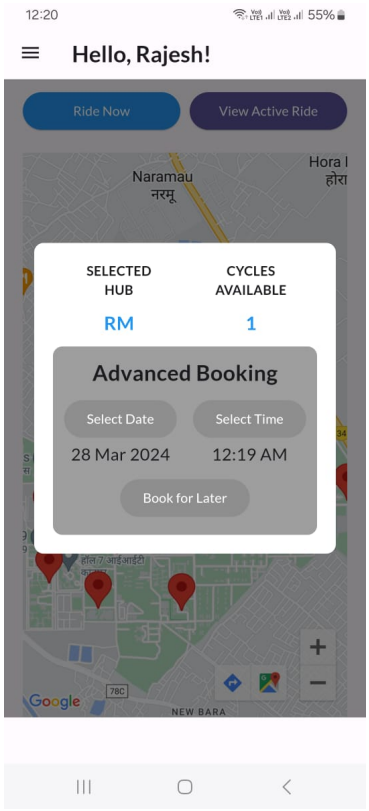
4.8 Advance Booking

Module Details: This module integrates the “Advance Booking” feature of the software.
Test Owner: Debraj Karmakar
Test Date: 27-03-2024
Test Results:

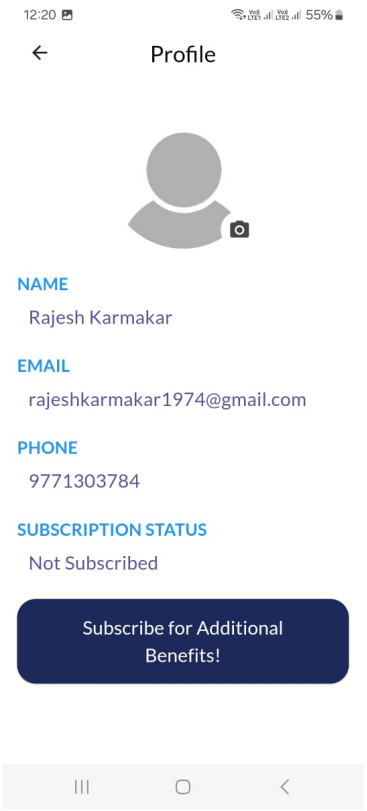
- **Test Case 1:** If user is an unsubscribed user.
  Result: Advance booking option is available for subscribed users only. The advance booking options are greyed out for unsubscribed users. Unsubscribed users can subscribe to PedalPal by clicking on the subscribe button in Profile section and then paying Rs.100 via the payment gateway.
- **Test Case 2:** If user is a subscribed user.
  Result: Subscribed users can book their rides in advance by paying an amount depending on the duration before which they are booking the cycles.They need to go to the advance booking section by clicking on the hub (on the map) where they want to book the cycle. Then select date and time and click on ‘Book for Later’ option and then pay the calculated amount.
- **Test Case 3:** If no free cycles are present at hub for advance booking.
  Result: On clicking on the hub, cycles available = 0. Also, the software prompts a message that “Hub does not have any available cycles!” and the booking process is cancelled.

- **Test Case 4:** Subscribed user does not have sufficient balance in wallet.
  Result: System prompts a message “Insufficient Balance” and the ride is not booked in advance.
- **Test Case 5:** If user books a cycle at a hub in advance and other user wants to unlock the cycle.
  Result: The software generates an error message “There was an error!” and cancels the advance booking process.

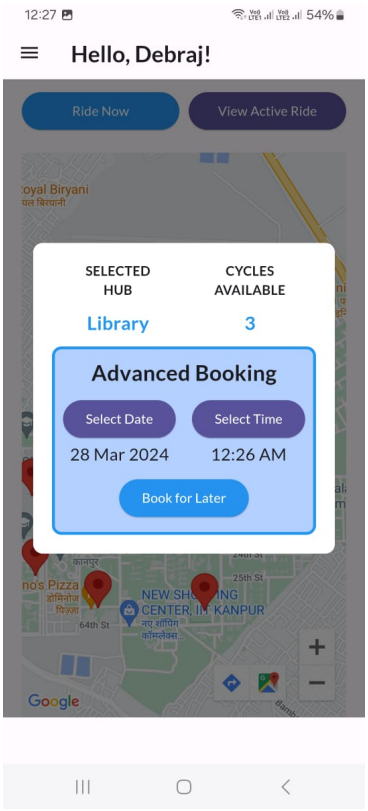
Additional Comments: None



(a) Greyed out advance booking for unsubscribed users

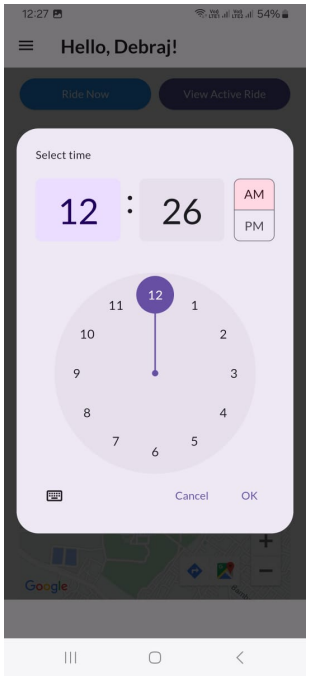


(b) Profile page with button for subscribing PedalPal

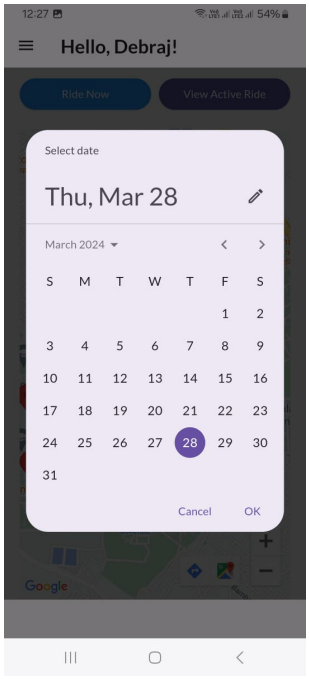


(c) Advanced booking interface for subscribed users

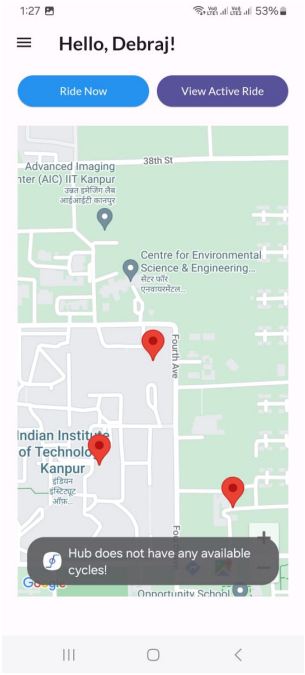
Figure 16: Advance Booking



(a) Set time interface

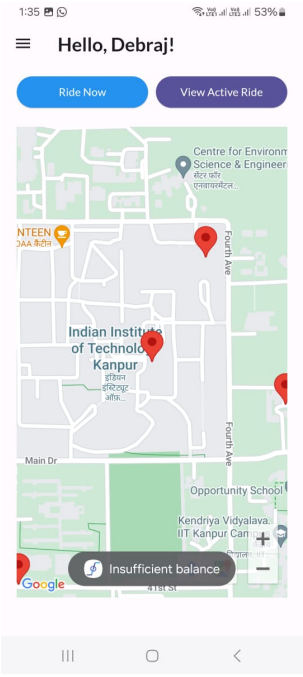


(b) Set date interface

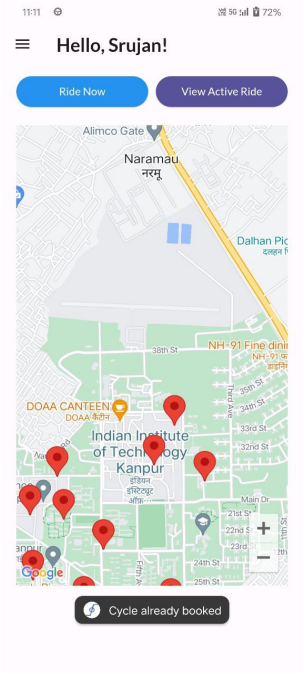


(c) Error: ”Hub does not have any available cycle”

Figure 17: Advance Booking



(a) Error: ”Insufficient Balance”



(b) Error when another person wants to book an already booked cycle

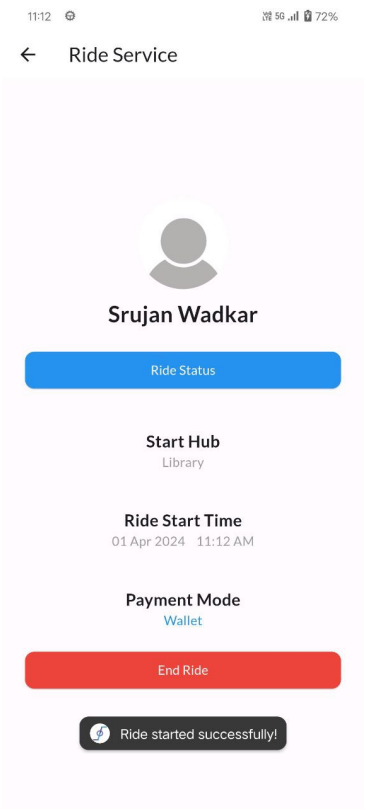
Figure 18: Advance Booking

4.9 View Active Rides

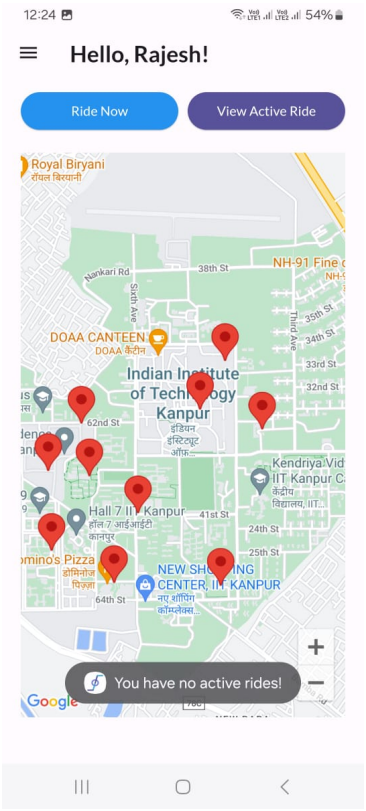
Module Details: This module displays the active rides of a user on the application.
Test Owner: Debraj Karmakar
Test Date: 27-03-2024
Test Results:

- **Test Case 1:** User has an active ride.
  Result: A user can book at most one cycle at a time. Hence, if the user has a active ride, its details can be viewed by clicking on the “View Active Ride” button on the home screen. Clicking on the ”View Active Ride” button redirects the user to the ’Ride Services’ page where details of the ride are displayed including the starting hub, starting time and payment mode.
- **Test Case 2:** User does not have an active ride.
  Result: In such a case, clicking the “View Active Ride” button prompts a message ‘You have no active rides!’.
- **Test Case 3:** “Ride Status” button in ’Ride Services’ page.
  Result: Clicking on the “Ride Status” button in View Active Ride displays ‘Ride Duration’ and ‘Current Amount’ which is calculated based on the present ride duration.

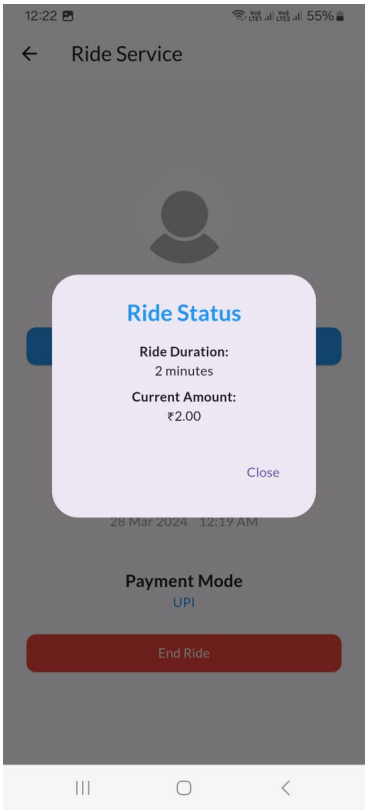
Additional Comments: None



(a) Ride Services Page



(b) Error: ”You have no active rides!”



(c) Ride status dialogue box

Figure 19: View Active Rides

5. System Testing

5.1 Functional Requirements

5.1.1 Launches Successfully

Requirement: App launches properly, all images and text are rendered correctly.
Test Owner: Raghav Manglik
Date: 30/03/2024
Test Results: App launched properly with all text and images rendered, along with a map of IIT Kanpur on Home page as expected.
Additional Comments: The site was launched on Android, development for iOS compatibility is pending.

5.1.2 Registration and Authentication

Requirement: App allows new users to register while enabling existing users to log in.
Test Owner: Raghav Manglik
Date: 30/03/2024
Test Results: Initially, the login functionality was tested using the credentials of existing website users. If registered users input correct details, log in was successful. Additionally, we tested the ”forgot password” feature and successfully logged in using the new password. Moreover, An unregistered email input in ”forgot password” feature was successfully recognised by the system. Further, we utilized the ”sign-up” feature to create a new user, which redirected the website to the login page. We then attempted to log in with the newly created user, and this functioned as expected. We also attempted to register a user using credentials that belonged to an already existing user. In this case, an alert prompt stating ”Profile with this email ID already exists” appeared.
Additional Comments: The current registration page’s full name input box requires users to input their name in the format of <First Name> <Surname>. We intend to modify this in future versions of the app.

5.1.3 Subscription

Requirement: App enables users to subscribe by making a one-time payment of Rs.20.
Test Owner: Raghav Manglik
Date: 30/03/2024
Test Results: Initially, attempts to access paid services through an unsubscribed account were tested. Services such as the In-App Wallet and Advance Bicycle Reservation were inaccessible to unregistered accounts. Subsequently, we attempted to subscribe using the option provided in the My Profile page. This button directed us to the Razorpay Payment Gateway, where Rs.100 was deducted from our account, and Rs.80 of it was credited to the activated Wallet. After subscribing, both the Wallet and Booking services became accessible.
Additional Comments: The subscription status on My Profile page was also updated from ”Unsubscribed” to ”Subscribed” following the payment of fee.

5.1.4 Renting and Returning Bicycles

Requirement: App facilitates the rental and return of bicycles.
Test Owner: Raghav Manglik
Date: 30/03/2024
Test Results: We attempted to rent a bicycle by selecting the ”Ride Now” option on the home page. Subsequently, a QR Scanner was displayed. Upon scanning a valid QR code (previously generated by us), the bicycle was successfully rented. Clicking on the ”View Active Ride” option on home page successfully allowed access to statistics for the current ride. Subsequently clicking on the ”End Ride” button opened a QR Scanner. Upon scanning a valid QR code, payment for the ride was required. For subscribed users, the fare was deducted from their wallet, while for unsubscribed users, they were directed to the Razorpay payment gateway. After successfully making the payment, the bicycle was returned successfully.
Additional Comments: Several edge cases were checked. If a subscribed user had a negative balance, No bicycle could be rented before clearing the dues. If an invalid QR code was scanned while renting or returning a bicycle, an error message appeared, providing a valid reason for the error. Also If a user tries to rent a bicycle during an active ride, an error message is displayed.

5.1.5 Submitting Feedback

Requirement: Users can submit feedback about their ride experience.
Test Owner: Srishti Chandra
Date: 30/03/2024
Test Results: After successfully returning a bicycle, A diaolog box asking for the user feedback appeared. We were successfully able to skip the feedback by tapping on ”Later” and were directed to Home page. Subsequently when we tapped on ”Sure” instead, We were directed to two windows, to report possible issues with the bicycle and a text description of it. We were successfully able to submit feedback and then redirect to Home Page.
Additional Comments: None

5.1.6 In-App Wallet Service

Requirement: An In-App Wallet service for users to pay their fare and booking fee.
Test Owner: Raghav Manglik
Date: 30/03/2024
Test Results: When an unsubscribed user attempts to access the wallet, an alert prompt notifies them that it is a subscribed service. For subscribed users, the wallet is displayed successfully, showing the correct balance along with options to add funds or view transaction history. We were able to successfully add funds to the wallet by selecting the ”Add Balance” option, entering a valid amount in the input box, and completing the payment through the Razorpay Payment gateway. Additionally, after any transaction, whether it’s adding funds or paying fare, the transaction log was accurately updated. Displaying different colors for different types of transactions, with red indicating debit transactions and green indicating credit transactions.
Additional Comments: None

5.1.7 Advance Booking of Bicycles

Requirement: App allows user to reserve bicycles for later usage.
Test Owner: Srishti Chandra
Date: 30/03/2024
Test Results: During our testing, we found that non-subscribed users were unable to book rides in advance. This feature is exclusively available for subscribers. When clicking on the hub location on the map, a prompt appeared allowing users to input the desired date and time for booking. If the user's balance was sufficient, the specific cycle was successfully booked; otherwise, an "Insufficient Balance" error message was displayed. Upon successful booking, clicking "Book for Later" deducted the appropriate amount from the user's wallet, and a confirmation message "Booking was Successful" was shown.
Additional Comments: None

5.1.8 Availability of Cycles at each Hub

Requirement: App facilitates the real-time availability of number of cycles at each hub.
Test Owner: Srishti Chandra
Date: 30/03/2024
Test Results: Home Page appears as expected with a map of IIT Kanpur with red pins pointing to the locations of Bicycle Hubs in the campus. Clicking on a red points correctly showed the number of cycles available at that hub. After A bicycle was booked, rented or returned by an user, this number was correctly updated.
Additional Comments: The map can be properly zoomed in or out using buttons or through manual adjustments.

5.1.9 Booking History

Requirement: App correctly displays the Booking logs of a user.
Test Owner: Srishti Chandra
Date: 30/03/2024
Test Results: After booking a bicycle, the Booking History page is correctly updated as bicycle is added in current bookings list. Following the expiration of the booking, the bicycle becomes available for rent/book as expected, and bicycle is transferred to past bookings list.
Additional Comments: None

5.1.10 Ride History

Requirement: App correctly displays the bicycle rental logs of a user.
Test Owner: Raghav Manglik
Date: 30/03/2024
Test Results: After returning a rented bicycle, the statistics of the ride, including starting and destination hub and duration of the ride are added to the Ride history log successfully.
Additional Comments: None.

5.1.11 User Profile and Logout

Requirement: Users can see their information and Logout.
Test Owner: Raghav Manglik

Date: 30/03/2024
Test Results: We were able to view correct Name, registered Email ID, Phone Number and Subscription status of the user at My Profile page. Logging out was also successful, as we were redirected to the ”Login” or ”Register” page.
Additional Comments: None

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

How Effective and exhaustive was the testing?

- Most of the unit tests had 100% code coverage and 100% branch coverage. Integration tests covered some major features such as user login, cycle rental and return, interaction among different views such as login, registration, Dashboard, Ride Services Page, History Page, Profile Page etc. Stress testing (a type of testing that is so harsh, it is expected to push the program to failure) was done at multiple concurrency levels and thread counts.
- Planned test cases in a well-designed manner, ensuring comprehensive coverage of all possible scenarios. The test cases were based on the application's functional requirements and design specifications, as well as on real-world usage scenarios. This helped to ensure that all critical areas of the application are tested thoroughly, and any potential issues are identified and resolved before the application is released.

Which components have not been tested adequately?

- Some of the components which involved mocking have not been tested using an automation suite.
- We haven't tested our required hardware components (RFID Arduino lock) completely.

What difficulties have you faced during testing?

- Large interconnection among different modules and views made it difficult to perform the integration testing.
- Lack of familiarity with testing tools: We faced difficulties in testing as we were not familiar with the various testing tools available. This led to a delay in the testing process, as we had to spend time researching and learning how to use them effectively.

How could the testing process be improved?

- Testing process could be improved by using automated testing for frontend, as we did for backend.
- Begin testing as early as possible in the development process to identify issues early on, allowing more time for fixing them before release.

A. Group Log

NOTE: The group activities during Testing were largely asynchronous, where we kept constant communication via our Discord server and WhatsApp group.

S.No	Date	Description
1	20/03/2024	Discussed about Testing process, decided the timeline and divided work among teammates
2	25/03/2024	Unit Testing started
3	26/03/2024	Integration Testing started
4	28/02/2024	Work on User Manual started
5	30/02/2024	System Testing started
6	01/04/2024	Test Document Final Compilation