Test Plan Document

For

Cave Country Canoes

Cheyenne Pierpont

Bailey Forbes

Brandon Maurer

Test Plan

1 Introduction

This document outlines the test plan for the mobile application being developed for Cave Country Canoes (CCC). It serves as a blueprint for the testing activities to ensure the application meets functional and non-functional requirements.

2 Business Background

- The owner of Cave Country Canoes has asked that we make them an app. It will have a digital map that they have created of the river on it as well as safety information. Every year they get more phone calls of paddlers being confused and scared and needing directions or in some cases emergency services. The customers will be able to click a button and enter their approximate or precise location that will be sent to CCC. It also can be forwarded to emergency services If needed. Making getting to them faster, safer, and less time consuming. This is the map that Cave Country Canoes has kindly provided us and will be used inside the app

3 Test Objectives

The objectives of our test cases are to ensure the system is running properly and to find any bugs we may have missed. Specifically;

- Front end is maneuverable and doesn't have any issues
    - All buttons work
    - All text is readable
    - No endless loading

    - Map can be seen and used
    - Reports can be filled out and sent
- Back end runs as expected
- Communication between system and database is working properly

4 Scope

Inclusions

The test plan encompasses testing all functional and non-functional aspects of the mobile application. The primary focus will be on ensuring the app's usability, performance, security, and functionality meet the requirements specified by Cave Country Canoes (CCC). Testing will also include the digital map's accuracy and the location-sharing features.

5 Test types Identified

- Functional Testing: Verifying that the app features work as expected.
- Usability Testing: Ensuring the app is easy to navigate and use.
- Performance Testing: Testing the app's speed, responsiveness, and stability.
- Security Testing: Verifying data safety, especially with location-sharing and reports.
- Compatibility Testing: Ensuring the app works seamlessly on various devices and operating systems.

- Integration Testing: Testing the communication between the app and the backend system.
- End-to-End (E2E) Testing: E2E testing is a methodology that ensures the complete flow of an application, from the front end to the back end, functions as intended. It validates the integration and interaction of different components, such as the user interface, backend services, and databases, simulating real-world user scenarios to ensure the system behaves as expected in actual use.

6 Problems Perceived

Possible problems we might encounter could be:

Ensuring the digital map loads correctly with minimal errors.

Managing variations in mobile device hardware and software.

Testing the app's ability to forward data accurately to emergency services.

Ensuring the system can handle simultaneous location reports from multiple users.

7 Architecture

Overall App Architecture

Each user type will have their own dashboard which includes:

1. Customers
2. Employees
3. Admins

4. Super Admins

Customers will be able to

1. Register an account through /signup_page
2. Enter in their customer portal through the /login_page -> routing to /customer_dashboard route. This also calls this API through /api/login.
3. From /customer_dashboard customers can:
4. Chat through /chat route.
5. Leave a review through /reviews_page route.
6. Fill out a ticket when they are in an emergency SOS situation in /submit_log_page route.
7. Logout which calls the API /api/logout
8. Customers can change account details by navigation of OTP verification.
9. Be able to send a OTP through 2FA verification via email allowing them to update their password if they forgot the password.

Employees will be able to do the following:

1. Login through /login_page -> /employee/home (dashboard)
2. Chat through /employee/chat
3. See reviews customers through /employee/see_reviews
4. See assigned emergencies that they have individually through the /employee/ individual_emergencies route. It is in this route, where the employee would mark the ticket as resolved, and the ticket would be deleted from the record.
5. See all emergencies through /employee/all_emergencies which will display all the emergency tickets/logs on record which are not resolved. This includes tickets

that have not been answered/assigned either by an employee or admin or are being resolved.

6. Employees can see reviews of the system via /employee/see_reviews route.
7. Employees can log out which calls upon the logout API.

Admins will be able to do the following:

1. Log in through the login route
2. Admins will be able to unassign or assign emergencies to employees if the employee account is not locked (see super admin section for more details).
3. Admins can make employee accounts, delete employee accounts,and update employee account details. This is done through the admin/manage_
4. Communicate in public chat through the admin chat route.
5. Logout through calling upon the API
6. Elevate their privileges to super admin by entering a designated root/master password (this is not the database root password).

Super Admins will be able to:

1. Do everything that a base admin can do.
2. Along with the ability to make employee accounts, super admins will be able to make admin accounts. This is done by the /admin_manage_staff route.
3. Inside the admin manage staff route super admins can lock accounts which prevents the now-locked account from being used. If the user is signed in, their next interaction will make them log out, and they cannot login again until unlock. Locking an account would commonly and should be used if security suspicion or the account user should not have access to the account (like vacation or leave).

4. Since it is assumed super admins are the only ones to know the master/root admin password, super admins are responsible for creating the first admin account. Creation of the first admin account requires the master admin password.

8 Environment

The testing environment will include:

Devices: A range of mobile devices (iOS and Android) with varying screen sizes and specifications.
Operating Systems: Different versions of iOS and Android.
Network Conditions: Testing under different network speeds and connectivity scenarios.

9 Assumptions

Once the system is pushed to the cloud, the person who sets up the first admin account must be a super admin through /admin_setup because this route requires the master/root admin password. This is required because Heroku wipes the database clean when uploaded to their servers.

We developers will do our best to reduce the chance of security flaws in the system, a cyber breach is not guaranteed.

10 Functionality

This system is designed to help Cave Country Canoes manage emergency reports through a ticket system. Emergencies are filed by customers on the customer

dashboard and carried out by employee's on the employee dashboard. Occasionally admins (and super admins) will manually assign emergencies to employees which are ticketed, so that employees can be notified.

An email bridge notifies employees and customers when an emergency ticket is answered, dropped or manually issued (or dropped) by an admin, including contact details. It also alerts account owners if their account is locked, dropping assigned emergencies for employees and notifying customers. Users are informed when an account is unlocked. The email bridge is also used for 2FA purposes when it comes to one-time passwords which is used for account detail changes.

Constraints and Resolutions

Parameter	**Customer Constraints**	**Infosys Limitations**
Internet Service	The Paddlers may or may not have internet throughout the entire trip depending on where they are on the river.	Certain cell phone services may not work at all so they may not be able to log if a phone call or text has or was being sent out.
Cellular Devices	Paddlers may or may not have their phones on them.	This depends on the person and situation. If they do not have their phone or they lose it down the river there is no way to send information unless they find other paddlers and use their devices.
Riverway App	Paddlers may decide they do not want to download	This is at the person's own choice; they may

	the app and use the functionality.	be able to download it once they are out there if they have the internet to do so..
Under the Influence	Paddlers may be incapable of using the app if they are under the influence while out there on the river.	This is at the person's own choice and doing.

Test Strategy

Unit tests, E2E tests, and integration tests will be used in order to test port behavior of 2XX, 3XX, 4XX, and 5XX when it comes to the website. The tests will be achieved through the use of the pytest library. The tests developed through the use of the pytest library will also provide a framework which will allow us to test the way data is transmitted to and from the database.

A rough testing file can be seen below. Note: SQLlite is used for testing purposes as it is a micro managed DB which can be easily overridden and separated from MySQL process.

```
import pytest
from app import app, db, Users, Emergencies, Ratings
from flask import session


# Fixture to initialize a test client and database
@pytest.fixture
def test_client():
    app.config['TESTING'] = True
    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///:memory:'  # Use
in-memory DB for tests
    app.config['WTF_CSRF_ENABLED'] = False  # Disable CSRF for testing
    with app.test_client() as client:
        with app.app_context():
            db.create_all()  # Create tables
        yield client
```

```
        with app.app_context():
            db.session.remove()
            db.drop_all()  # Clean up tables after tests


# Fixture for creating test data
@pytest.fixture
def setup_test_data():
    with app.app_context():
        # Create a test user
        user = Users(
            username="testuser",
            email="testuser@example.com",
            password_hash="hashedpassword",
            phone_number="1234567890",
            account_type="customer"
        )
        db.session.add(user)

        # Create an employee
        employee = Users(
            username="testemployee",
            email="testemployee@example.com",
            password_hash="hashedpassword",
            phone_number="0987654321",
            account_type="employee"
        )
        db.session.add(employee)

        # Create an admin
        admin = Users(
            username="testadmin",
            email="testadmin@example.com",
            password_hash="hashedpassword",
            phone_number="1122334455",
            account_type="admin"
        )
        db.session.add(admin)

        # Create an emergency
        emergency = Emergencies(
            user_id=1,
            location_details="Test Location",
            distress_notes="Test distress notes",
            assigned_employee_id=None
        )
```

```
        db.session.add(emergency)

        # Create a rating
        rating = Ratings(
            user_id=1,
            rating_header="Great service",
            rating_notes="Everything went smoothly.",
            rating_value=5
        )
        db.session.add(rating)

        db.session.commit()



############################
#      ROUTE TESTS       #
############################

def test_index_redirect(test_client):
    """Test: Index route should redirect to login page."""
    response = test_client.get("/")
    assert response.status_code == 302
    assert response.location.endswith("/login_page")


def test_login_page_rendering(test_client):
    """Test: Login page should render successfully."""
    response = test_client.get("/login_page")
    assert response.status_code == 200
    assert b"Log In" in response.data


def test_valid_login(test_client, setup_test_data):
    """Test: Valid login for customer."""
    response = test_client.post("/login_page", data={
        "email": "testuser@example.com",
        "password": "hashedpassword"
    }, follow_redirects=True)

    assert response.status_code == 200
    assert b"Logged in successfully!" in response.data


def test_invalid_login(test_client):
    """Test: Invalid login credentials."""
```

```
    response = test_client.post("/login_page", data={
        "email": "wronguser@example.com",
        "password": "wrongpassword"
    }, follow_redirects=True)

    assert response.status_code == 200
    assert b"Invalid email or password" in response.data

def test_employee_home_unauthorized(test_client):
    """Test: Accessing employee home without logging in should redirect."""
    response = test_client.get("/employee/home", follow_redirects=True)
    assert response.status_code == 200
    assert b"Please log in" in response.data


def test_employee_home_authorized(test_client, setup_test_data):
    """Test: Employee home page access with valid login."""
    with test_client:
        test_client.post("/login_page", data={
            "email": "testemployee@example.com",
            "password": "hashedpassword"
        }, follow_redirects=True)
        response = test_client.get("/employee/home")
        assert response.status_code == 200
        assert b"Welcome, testemployee" in response.data


def test_admin_home_authorized(test_client, setup_test_data):
    """Test: Admin home page access with valid login."""
    with test_client:
        test_client.post("/login_page", data={
            "email": "testadmin@example.com",
            "password": "hashedpassword"
        }, follow_redirects=True)
        response = test_client.get("/admin/home")
        assert response.status_code == 200
        assert b"Admin Dashboard" in response.data


############################
#    DATABASE TESTS    #
############################

def test_user_creation(test_client):
    """Test: User creation in database."""
    with app.app_context():
```

```
        user = Users(
            username="newuser",
            email="newuser@example.com",
            password_hash="hashedpassword",
            phone_number="9876543210",
            account_type="customer"
        )
        db.session.add(user)
        db.session.commit()

        fetched_user = Users.query.filter_by(email="newuser@example.com").first()
        assert fetched_user is not None
        assert fetched_user.username == "newuser"


def test_emergency_creation(test_client, setup_test_data):
    """Test: Emergency creation in database."""
    with app.app_context():
        emergency = Emergencies(
            user_id=1,
            location_details="New Location",
            distress_notes="Help required"
        )
        db.session.add(emergency)
        db.session.commit()

        fetched_emergency = Emergencies.query.filter_by(location_details="New
Location").first()
        assert fetched_emergency is not None
        assert fetched_emergency.distress_notes == "Help required"


def test_rating_creation(test_client, setup_test_data):
    """Test: Rating creation in database."""
    with app.app_context():
        rating = Ratings(
            user_id=1,
            rating_header="Good service",
            rating_notes="Quick and reliable",
            rating_value=4
        )
        db.session.add(rating)
        db.session.commit()

        fetched_rating = Ratings.query.filter_by(rating_header="Good service").first()
        assert fetched_rating is not None
```

```
        assert fetched_rating.rating_value == 4



############################
#   API ENDPOINT TESTS   #
############################

def test_api_signup(test_client):
    """Test: API signup."""
    response = test_client.post("/api/signup", json={
        "username": "apiuser",
        "email": "apiuser@example.com",
        "password": "testpassword",
        "phone_number": "1234567890"
    })
    assert response.status_code == 201
    assert b"Account created successfully!" in response.data


def test_api_login(test_client, setup_test_data):
    """Test: API login."""
    response = test_client.post("/api/login", json={
        "email": "testuser@example.com",
        "password": "hashedpassword"
    })
    assert response.status_code == 200
    assert b"Logged in successfully!" in response.data


def test_api_emergency_post(test_client, setup_test_data):
    """Test: API to create an emergency."""
    with test_client:
        test_client.post("/login_page", data={
            "email": "testuser@example.com",
            "password": "hashedpassword"
        })
        response = test_client.post("/api/emergency", json={
            "location_details": "Emergency Location",
            "distress_notes": "Emergency distress"
        })
        assert response.status_code == 201
        assert b"Emergency log submitted." in response.data
```

Automation Plans

The email bridge will need to be tested as it serves as an important tool for automation and communication. Tools such as Selenium can be used for automated testing. A rough example of Selenium testing via python which is relative to the project:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time


# Path to ChromeDriver
CHROMEDRIVER_PATH = "/path/to/chromedriver"

# Test URLs
BASE_URL = "http://127.0.0.1:5000"


def test_login():
    """Test the login functionality using Selenium."""
    # Initialize WebDriver
    service = Service(CHROMEDRIVER_PATH)
    driver = webdriver.Chrome(service=service)
    driver.get(f"{BASE_URL}/login_page")

    try:
        # Locate and fill the email field
        email_field = driver.find_element(By.NAME, "email")
        email_field.send_keys("testuser@example.com")

        # Locate and fill the password field
        password_field = driver.find_element(By.NAME, "password")
        password_field.send_keys("hashedpassword")

        # Submit the form
        password_field.send_keys(Keys.RETURN)

        # Wait for the dashboard page to load
        WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.TAG_NAME, "h1"))
        )

        # Validate success message or redirection
```

```
        assert "Dashboard" in driver.page_source
        print("Login test passed!")
    except Exception as e:
        print(f"Login test failed: {e}")
    finally:
        # Close the browser
        driver.quit()
def test_emergency_submission():
    """Test emergency log submission."""
    # Initialize WebDriver
    service = Service(CHROMEDRIVER_PATH)
    driver = webdriver.Chrome(service=service)
    driver.get(f"{BASE_URL}/login_page")
    try:
        # Login first
        driver.find_element(By.NAME, "email").send_keys("testuser@example.com")
        driver.find_element(By.NAME, "password").send_keys("hashedpassword",
Keys.RETURN)
        WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.LINK_TEXT, "Submit Emergency"))
        )

        # Navigate to emergency submission page
        driver.find_element(By.LINK_TEXT, "Submit Emergency").click()

        # Fill in emergency details
        driver.find_element(By.NAME, "location_details").send_keys("Test Location")
        driver.find_element(By.NAME, "distress_notes").send_keys("Test distress")

        # Submit the form
        driver.find_element(By.XPATH, "//button[contains(text(), 'Submit')]").click()

        # Validate success message
        success_message = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.CLASS_NAME, "alert-success"))
        )
        assert "Emergency log submitted successfully!" in success_message.text
        print("Emergency submission test passed!")
    except Exception as e:
        print(f"Emergency submission test failed: {e}")
    finally:
        # Close the browser
        driver.quit()
if __name__ == "__main__":
    test_login()
    test_emergency_submission()
```

Deliverables

11 Security

Constraints and Resolutions

Parameter	**Customer Constraints**	**Infosys Limitations**
XSS	Cross-Site Scripting(XSS) attacks are type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.	An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can rewrite the content of the HTML page.
SQL Injection	SQL Injection occurs when an attacker is able to	● unauthorized access to sensitive data: attackers can

	manipulate a web application's database queries by inserting malicious SQL code into user input fields. These injected queries can manipulate the underlying database to retrieve, modify, or delete sensitive data or a server.	retrieve personal, financial, or confidential information stored in the database. ● Data Integrity issues: Attackers can modify, delete, or corrupt critical data, impacting the applications functionality ● Privilege escalation: Attackers can bypass authentication mechanisms and gain administrative privileges. ● Service Downtime: SQL Injection can overload the server, causing performance degradation or system crashes. ● Reputation damage: A successful attack can severely harm the reputation of an organization, leading to a loss of customer trust. .
CSRF	ability to send commands to a website from an attacker to the server. Unlike XSS, CSRF exploits the trust between the site and the user.	

Session (token) management	Session token -> an authentication mechanism which is given to entrusted users once the user has completed an expected task	
DDoS	A type of DoS which uses a network of servers or computers connected to perform a DoS operation.	● Similar outcomes to a DoS
DoS	The use of a program on a single device or server which will send multitudes of requests resulting in inability to use a website or other service.	● An attacker can use a DoS to take down a website. If a DoS is successful then there is nothing really one can do. This can result in financial loss and liability.

One crucial factor to consider is the security of this product. This encompasses both the functionality of the front end and its aesthetic design. A specific example of security that needs to be addressed is SQL injection. To protect a website from SQL injection attacks, the front end code needs to be modified, for instance, by using parameterized queries or similar methods.

Risk Identified & Mitigation Planned

- XSS - ability to add unauthorized scripts or "inject" code into middleware.
    - Code front end input to reduce scripting and take in and process text as string literals.
- SQL Injection - The ability to use SQL to elevate permissions and bypass credentials.

    - Use input authentication and checks such as when an email is requested, make sure that input is an email. Ensure middleware does not leak potentially dangerous SQL exploits. Use tools such as regexs for text checking.
- CSRF - Ability to send commands to a website from an attacker to the server. Unlike XSS, CSRF exploits the trust between the site and the user.
    - Use session management and tokens.
- Session token - An authentication mechanism which is given to entrusted users once the user has completed an expected task.
    - Use rotating tokens and ensure that the tokens are stored in user tables so that the token can be tied to the user_id through foreign keys.
- DoS - The use of a program on a single device or server which will send multitudes of requests resulting in inability to use a website or other service.
- DDos - A type of DoS which uses a network of servers or computers connected to perform a DoS operation.
    - Use tools such as Cloudflare for DoS and DDoS protection and insurance.

12 Performance

Constraints and Resolutions

Parameter	**Customer Constraints**	**Infosys Limitations**
App Being Down	Employees and Paddlers will not be able to access any of the information that is held in the app.	May not be able to access the information that is currently stored. Will not be able to take in any new data that could possibly be sent in if paddlers are able to communicate when the app is down.
No Internet Connection	Employees and Paddlers will not be able to access any of the information that is held in the app.	May not be able to access the information that is currently stored. Will not be able to take in any new data that could possibly be sent in if

		paddlers are able to communicate when the app is down.
No Cellular Device	Anyone without a device will not have access to any of the information. Unless being sent through another paddlers device.	Anyone without a device will not have access to any of the information. Unless being sent through another paddlers device.

13 Usability

Super admins have the most freedom, accessing both the back and front end, cloud space, and managing the email bridge account. They can create basic admins along with, manage existing admin accounts; note: it is recommended that they privately share the escalation password, which grants super admin privileges. Super admins can lock accounts for security or access reasons and perform all admin tasks due to their escalated privileges - super admins are admins themselves.

Admins will be able to make employee accounts and manage employee accounts. Admins will be able to manually assign emergency tickets to employees in the event a ticket is unanswered (the customer who made the ticket and the employee who is now assigned are notified via email). Admins can chat inside the chat room.

Employees can view reviews assigned by admins or self-assigned on this page. They can certify completed tickets and see all emergency tickets, including those answered by others or not. A third page displays customer reviews with a five-star rating, header, description, and username. Employees can also chat.

Customers will be able to fill out emergency report tickets, change their account info, make a review, and chat. These options will be on their own separate routes. The ability to change account info for customers will use OTP verification through the email bridge and goes through a three-route process.

Constraints and Resolutions

Parameter	**Customer Constraints**	**Infosys Limitations**
Super Admin	Only a few people will be able to help in certain situations. If there is an issue and a super admin is not around could delay in response time.	Paddlers may not get a response as soon as they'd like. if there is an issue with an account or the information being sent in.
Admin	If there is a problem and they do not have clearance they will have to explain why the issue can not be resolved at the current time. Will have to reach out and get a hold of a super admin.	Paddlers may not get a response as soon as theyd like. if there is an issue with an account or the information being sent in.
Employee	If there is a problem and they do not have clearance they will have to explain why the issue can not be resolved at the current time. Will have to reach out and get a hold of a super admin.	Paddlers may not get a response as soon as they'd like. if there is an issue with an account or the information being sent in.

14 Test Team Organization

There are only the three of us developing the application, so we all will test it on our own devices. We could possibly contact others with different devices so we can test it on their devices as well.

15 Schedule

Back end will be first developed with some front end to test the back end. This includes routes and the API. Once the back end and a basic front end is built, the front end will be polished for a production quality. After both are done the main security and the main testing phase will begin. As a note: security and testing will be done throughout.

Once the system is done, we will upload the process to the cloud through the use of Heroku. When on the cloud we will do a last stage of testing to make sure that the system works correctly in the cloud.

When we are confident that the system is properly functioning on the cloud, we will present the project to the client. A trial phase will begin on the system for both Cave Country Canoes staff and the customers of Cave Country Canoes. During this trial we will act as consultants for staff and customers until the staff themselves feel comfortable with the system.

16 Defects Classification Mechanism

Type of Defects	Functionality	Performance	Security	Usability	Compatibility
Critical	x		x		x
Major				x	
Minor		x			
Cosmetics	x		x	x	

One major factor which will need to be considered is the security of this product. This factors in with functionality of the front end, and regarding its cosmetic design. A specific example of security which will have to be considered is Cross Site Scripting (XSS). To XSS proof a website, the front end will need to be written differently such as using .innerText() instead of .innerHTML() or similar.

Speed and performance are a minimal requirement due to Flask's lightweight nature. Usability can differ when it comes to cosmetics, but it should still be usable by several input types.

Compatibility and usability are a must because the system will need to be supported for mobile device input (ability to take finger input) primarily and desktop input (keyboard and mouse). As search the front end will have to focus on a touch friendly design.

Turn Around Time for defect fixes

It is important to detect bugs and fix them before continuing with development. As we develop concepts it is our plan to test a new overall (single) concept for bugs before it is fully integrated.

17 Configuration Management

Version control will be managed using Git to track changes in the source code and test cases.
All test cases and test data will be documented and stored in a central repository.

18 Release Criteria

See section 15