Prototype Test Plan

Project Group 10 - November 3, 2024

Objective: Prototype I, establishing foundational functionality for critical subsystems, including **Restricted Zone Access or Zone Restricted Access (ZRA)** and **Location Acquisition**. Additionally, developing and updating the **Bill of Materials (BOM)**, refine prototype plan, and initiate testing to validate key performance metrics..

Define Measurements

- API - Table 1
- Zone Restricted Access (ZRA) - Table 2

Table 1: API Defined Measurements

Lines of code	77
Coordinates of user (X,Y)	(**Dynamic**; values vary with user location)
Coordinates of Zone (X,Y)	(**Dynamic**; values depend on designated zone)

Table 1: ZRA Defined Measurements

Lines of code	26
Authorization value	$X \in \Re$ - Specific to access level for each zone

---

Bill of Materials

Item	Description	Quantity	Unit Cost ($)	Total Cost ($)	Links
Shabodi's APIs	Access to Shabodi's APIs for location, and other network capabilities	1	0	0	Shabodi Sandbox
Python Development Environment - VSCode	Open-source software for backend development and API integration	1	0	0	N/A
Database Software - Firebase Specifically Gigaspaces or MongoDSB	Open-source database software	1	0	0	Understanding your Data Stack Handbook - Gated (gigaspaces.com)
Wi-Fi/Bluetooth Testing Tools	Open-source tools (scapy) for device detection and network control scripting (psutil)	1	0	0	Scapy - Introduction — Scapy 2.6.0 documentation Psutil - psutil · PyPI
Development Tools - Webapp built with React	Open-source tools for developing an optional user interface	1	0	0	React

Testing Objectives and Scope

Primary Testing Focus: For this prototype, testing will concentrate on two essential subsystems

1. **Restricted Zone Access (ZRA) -** Ensuring that only authorized users can enter designated areas
2. **Location Acquisition -** Accurately capturing user coordinates for zone validation

By validating these components in isolation, we establish a solid foundation for further development and integration in future prototypes.

API Analysis:

- **Zone-Specific Requirements**: AUTHORIZED_ZONES is a dictionary where each zone has a location, radius, and required authority level.

- **Authority Level Check**: In restricted_zone_access, the has_access function compares the user's authority level with the zone's requirement before proceeding with location validation.

- **Access Denial for Low Authority**: If the authority level is too low, access is denied, and the superior is not notified.

Zone Restricted Access: Analysis

- **Authorization Check**: has_access verifies if the user's authority_level meets or exceeds the required level for the specified zone.
- **Access Message**: Based on the result, restricted_zone_access outputs either "Access Granted" or "Access Restricted."

Testing Procedure

Running the ZRA Code

The ZRA functionality can be tested by inputting a user's authority level and the zone name. If the entered level satisfies the zone's requirement, "Access Granted" is displayed. Otherwise, it will return "Access Restricted."

Running the API Code

The API's code currently serves as a template, pending access to Shabodi's proprietary API. When access is granted, API fields will be populated for full testing capability,

Prototype 1 - Couldn't access API, but functional in current state as it pulls necessary data and completes analysis to determine location

Design Concept	Reason for Prototype	Describe the Concept to be Prototyped and Tested	Level of Prototype	Kind of Prototype	Metrics	Test Description	Analysis Method	Results	Interpretation	Notes
Prototype 1 - Proof of Concept	Device Detection and Logging	Demonstrate basic device detection and ability to log devices within specified zones using addresses	Focus	Analytical	Device detection accuracy: 80% or higher over a 30-minute interval	Scan for devices within a specified Wi-Fi range	Monitor device presence within the zone and log data in the database; verify accuracy by comparing manuel logs with system logs	At least 80% detection accuracy in device logging	Pass if device detection is accurate 80% of the time or higher over a 30-minute testing interval	Testing conducted using controlled Wi-Fi network setup to ensure signal consistency

Prototype 2 - Critical Subsystem Testing	Access Restriction and Bandwidth Control	Verify access restriction and bandwidth control capabilities using Shabodi's API	Focus	Analytical	Response time for access restriction: 2 seconds or less	Set up restricted zones and apply bandwidth limits when a device moves out of bounds	Measure response time by timing delay between device detection out of bounds and restriction activation	Restriction applied within 2 seconds of device detection out of zone	Pass if the system limits access within 2 seconds and maintains restriction consistently	Testing conducted in different zones to verify response stability across varied network conditions
Prototype 3 - Database and Zone Management	Data Management for Zones and Devices	Test storage and retrieval of device data in a database, as well as setting up, updating, and removing zones.	Focus	Analytical	Database retrieval time <1 second; accurate zone creation and updates	Test storing device data (MAC Address zones) and updating zone settings in database, ensuring data is quickly accessible and up-to-date	Query the database and measure retrieval times; validate zone accuracy by checking stored vs. displayed data	Database retrieval in <1 second and correct zone updates	Pass if retrieval time is under 1 second, and zone management functions (add, update, delete) work as expected	Testing includes adding multiple devices and zones to stimulate actual usage
Prototype 4 - Full System Integration	User Interaction and System Functionality	Test full system integration, including backend, API, database, and optional user interface functionality for monitoring	HiFi Comprehensive	Physical and analytical	System operates without crashes; UI Displays data within 2 seconds	Test complete system with all integrated components, allowing device management via optional UI and ensuring real-time updates	Interact with the system through the UI (if implemented) and validate end-to-end functionality for different user scenarios	System stable and UI updates within 2 seconds	Pass if system operates as intended with no crashes, and all components work in sync	Testing include various device entries/exits and user actions via the UI functionality and stability

Code/Prototype:

ZRA: A sample of the Zone Restricted Access code for initial testing is provided below

```
# Define zones with required authority levels for access
AUTHORIZED_ZONES = {
  "Zone_A": {"required_level": 5},
  "Zone_B": {"required_level": 7},
}

# Function to check if user meets the authority requirement for the zone
def has_access(authority_level, zone_name):
  zone_info = AUTHORIZED_ZONES.get(zone_name)
  if zone_info is None:
    print("Zone not found.")
    return False
  return authority_level >= zone_info["required_level"]

# Main function to check access for a specific zone
def restricted_zone_access(authority_level, zone_name):
  if has_access(authority_level, zone_name):
    print("Access Granted.")
  else:
    print("Access Restricted.")
```

```
# Example usage
authority_level = int(input("Enter user authority level: "))  # User's authority level
zone_name = input("Enter zone name (e.g., Zone_A): ")  # Name of the zone

restricted_zone_access(authority_level, zone_name)
```

API: Sample of the API code for initial testing is provided below:

```
\import requests
from datetime import datetime

# Constants (fill in according to Shabodi's API details)
SHABODI_API_BASE_URL = "https://api.shabodi.com/v1"  # Example base URL
API_KEY = "your_api_key_here"  # Replace with your actual API key

# Define zones and required authority levels
AUTHORIZED_ZONES = {
  "Zone_A": {"latitude": 40.712776, "longitude": -74.005974, "radius": 50, "required_level":
5},
  "Zone_B": {"latitude": 34.052235, "longitude": -118.243683, "radius": 50,
"required_level": 7},
}

# Function to get user's location
def get_user_location(user_id):
  url = f"{SHABODI_API_BASE_URL}/getLocation"
  headers = {"Authorization": f"Bearer {API_KEY}"}
  params = {"userId": user_id}
  response = requests.get(url, headers=headers, params=params)

  if response.status_code == 200:
    return response.json()  # Assuming response has latitude and longitude
  else:
    print(f"Error: {response.status_code}")
    return None

# Function to check if user meets the authority requirement for the zone
def has_access(authority_level, zone_info):
  return authority_level >= zone_info["required_level"]
```

```
# Function to check if user is within authorized zone
def is_in_zone(user_location, zone_info):
  # Here, use the Haversine formula or geolocation library to check distance
  lat, lon = user_location["latitude"], user_location["longitude"]
  distance = calculate_distance(lat, lon, zone_info["latitude"], zone_info["longitude"])
  return distance <= zone_info["radius"]

# Function to notify superior
def notify_superior(user_id, status):
  url = f"{SHABODI_API_BASE_URL}/notify"
  headers = {"Authorization": f"Bearer {API_KEY}"}
  data = {
    "userId": user_id,
    "status": status,
    "timestamp": datetime.now().isoformat()
  }
  response = requests.post(url, headers=headers, json=data)

  if response.status_code == 200:
    print("Superior notified successfully.")
  else:
    print(f"Notification failed: {response.status_code}")

# Main function to check access for each authorized zone
def restricted_zone_access(user_id, authority_level, zone_name):
  user_location = get_user_location(user_id)

  if user_location and zone_name in AUTHORIZED_ZONES:
    zone_info = AUTHORIZED_ZONES[zone_name]

    # Check if the user has the required authority level
    if has_access(authority_level, zone_info):
      # Check if user is within the zone
```

```
        if is_in_zone(user_location, zone_info):
            print(f"Access granted to {zone_name}.")
            notify_superior(user_id, "Arrived on time")
        else:
            print(f"User is not within the authorized zone for {zone_name}.")
    else:
        print(f"Access denied: insufficient authority level for {zone_name}.")
else:
    print("Access denied due to insufficient authority level or invalid zone.")
```

```
# Example usage
user_id = "user123"  # Replace with actual user ID
authority_level = 5  # Example authority level
restricted_zone_access(user_id, authority_level, "Zone_A")
```

Combined Code: Sample of the code integrated together for testing

```
import requests
from datetime import datetime
from math import radians, sin, cos, sqrt, atan2

# Constants (replace these with actual API details)
SHABODI_API_BASE_URL = "https://api.shabodi.com/v1"
API_KEY = "your_api_key_here"

AUTHORIZED_ZONES = {
    "Zone_A": {"latitude": 40.712776, "longitude": -74.005974, "radius": 50, "required_level": 5},
    "Zone_B": {"latitude": 34.052235, "longitude": -118.243683, "radius": 50, "required_level": 7},
}

def get_user_location(user_id):
```

```
    url = f"{SHABODI_API_BASE_URL}/getLocation"
    headers = {"Authorization": f"Bearer {API_KEY}"}
    params = {"userId": user_id}
    response = requests.get(url, headers=headers, params=params)

    if response.status_code == 200:
        return response.json()  # Assumes response contains latitude and longitude
    else:
        print(f"Error: {response.status_code}")
        return None

def has_access(authority_level, zone_info):
    return authority_level >= zone_info["required_level"]

def calculate_distance(lat1, lon1, lat2, lon2):
    # Haversine formula to calculate the distance between two points on Earth
    R = 6371.0  # Earth radius in km
    dlat = radians(lat2 - lat1)
    dlon = radians(lon2 - lon1)
    a = sin(dlat / 2)**2 + cos(radians(lat1)) * cos(radians(lat2)) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    distance = R * c * 1000  # Convert to meters
    return distance

def is_in_zone(user_location, zone_info):
    lat, lon = user_location["latitude"], user_location["longitude"]
    distance = calculate_distance(lat, lon, zone_info["latitude"], zone_info["longitude"])
    return distance <= zone_info["radius"]

def notify_superior(user_id, status):
    url = f"{SHABODI_API_BASE_URL}/notify"
    headers = {"Authorization": f"Bearer {API_KEY}"}
    data = {
        "userId": user_id,
```

```
        "status": status,
        "timestamp": datetime.now().isoformat()
    }
    response = requests.post(url, headers=headers, json=data)

    if response.status_code == 200:
        print("Superior notified successfully.")
    else:
        print(f"Notification failed: {response.status_code}")

def restricted_zone_access(user_id, authority_level, zone_name):
    user_location = get_user_location(user_id)
    if user_location and zone_name in AUTHORIZED_ZONES:
        zone_info = AUTHORIZED_ZONES[zone_name]

        if has_access(authority_level, zone_info):
            if is_in_zone(user_location, zone_info):
                print(f"Access granted to {zone_name}.")
                notify_superior(user_id, "Access granted and within zone.")
            else:
                print(f"User is not within the authorized zone for {zone_name}.")
                notify_superior(user_id, "Access granted but out of zone.")
        else:
            print(f"Access denied: insufficient authority level for {zone_name}.")
            notify_superior(user_id, "Access restricted due to low authority.")
    else:
        print("Access denied due to insufficient authority level or invalid zone.")
```

```
# Example usage
user_id = "user_123"
authority_level = 5
restricted_zone_access(user_id, authority_level, "Zone_A")
```

Critical Components Table

Component	Description	Purpose	Critical Functionality
Backend API (Shabodi)	Custom API for authority validation and access logging (as well as location detection through network)	Process requests, validates user authority, logs access events	Serves as the main hub for all system requests and logic
Database Service - Firebase Realtime	Cloud database	Stores zone definitions, user roles, and access logs	Enables fast retrieval and updating of access records
Access Control Logic	Functions for user validation and zone checking	Validates user access based on authority and zone parameters	Core functionality for access granting or restriction
Location Processing	Haversine distance calculation	Calculates the distance between user's location and zone	Determines if a user is within the allowed zone radius
Notification Service - Firebase Cloud Messaging	Push notification API	Sends alerts to supervisors on access attempts and outcomes	Keeps supervisors informed of access status in real time
Authentication Service	OAuth or JWT-based user authentication	Verifies user identity for secure access	Ensures only authenticated users interact with the system
Error Handling and Logging	Comprehensive error handling and logging services	Captures and logs error, stores activity history	Aids in debugging and provides a trail of access attempts
Mobile/Web Interface (Optional)	Simple UI for supervisors to view access logs and statuses	Allows for easy monitoring and control	User-friendly interface for real-time monitoring of access events

Test Data Table for Combined Code

Purpose: Conduct a comprehensive check of the ZRA system, including authority validation, location proximity, and supervisor notification. This test verifies that all integrated system components. This test verifies that all integrated system components work in harmony to grant or restrict access appropriately.

Test Case ID	User ID	Authority Level	Zone Name	Expected Latitude	Expected Longitude	Expected Access	Expected Notification
TC-01	user_001	5	Zone_A	40.712776	-74.005974	Granted	“Access granted and within zone.”
TC-02	user_002	3	Zone_A	40.712776	-74.005974	Denied	“Access granted due to low authority ”
TC-03	user_003	8	Zone_B	34.052235	-118.243683	Granted	“Access granted and within zone”
TC-04	user_004	7	Zone_B	34.05223	-118.243	Granted	“Access

				5	683		granted and within zone”
TC-05	user_005	5	Zone_A	34.000000	-118.243683	Denied	“Access restricted due to out-of-zone.”
TC-06	user_006	4	Zone_B	34.052235	-118.243683	Denied	“Access restricted due to low authority ”

Testing Overview:

Each component will be tested and evaluated independently, followed by integration testing to confirm all parts work together seamlessly.

Refined Prototyping Plan with Focused Subtests

This includes detailed subtests for individual components of the system:

Current Focus:

1. **Subtest A - Authority Validation**
    - **Objective:** Verify that the system correctly restricts access based on user authority levels for each zone
    - **Test Metric:** Authority check should grant or restrict access as specified in AUTHORIZED_ZONES

- **Pass Criterion:** User access matches expected access levels based on authority

2. **Subtest B - Location Proximity Validation**
    - **Objective:** Ensure accurate distance calculation between user and zone centre using the Haversine formula
    - **Test Metric:** Calculated distance aligns within a plus/minus 5% margin from expected
    - **Pass Criterion:** System consistently verifies user location is within defined zone radius

Next Focus:

3. **Subtest C - Notification Service**
    - **Objective:** Confirm notifications are sent to supervisors with correct status
    - **Test Metric:** Notification status and message match expected for given authority and zone proximity
    - **Pass Criterion:** Notification matches expected message for each user's access outcome

4. **Subtest D - Integrated Access Check**
    - **Objective:** Test full access validation including authority, location, and notification
    - **Test Metric:** Integrated function should output correct access status and notification message
    - **Pass Criterion:** Combined test outputs expected result based on authority, location, and notification tests

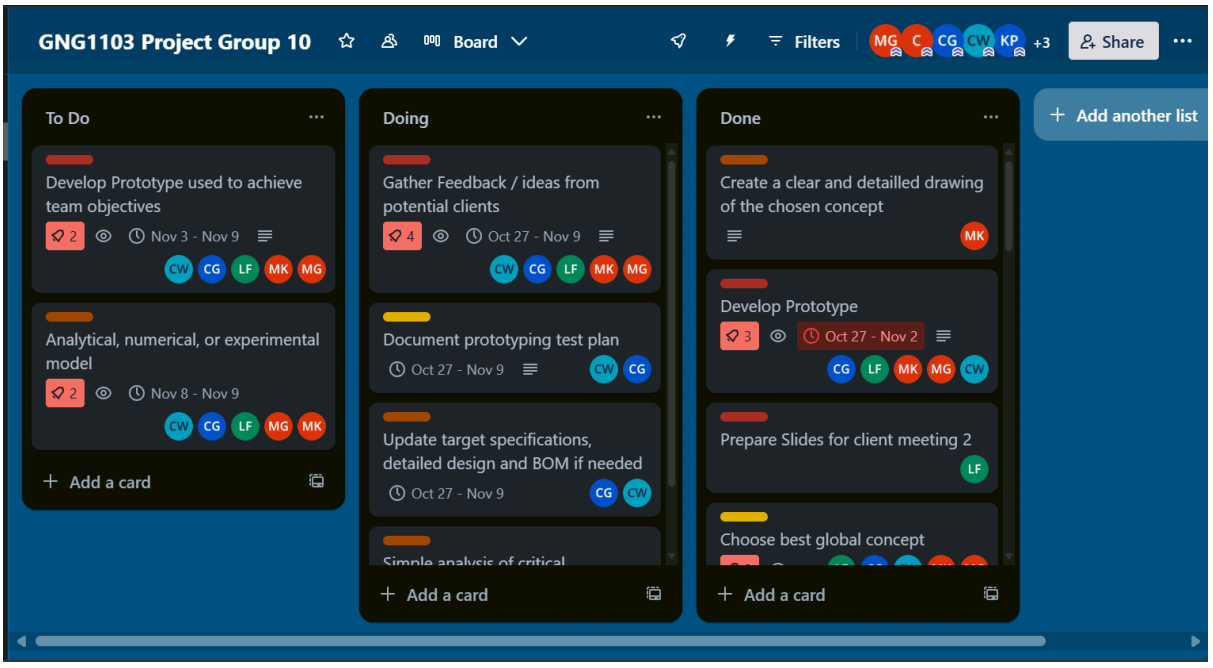
---

Feedback Notes from Potential Users and Others:

- Main control centre everything can be controlled and monitor from vs. cluttered email alerts
- Preference towards management through telephone, easy and simplistic
- Shabodi Feedback: Functionality over UI

Additional Notes:

- **Environmental Factors:** All tests should be conducted in a controlled setting with stable Wi-Fi for accurate location data
- **Testing Tools:** A location simulation tool will help verify location proximity without physical movement, ensuring precision
- **Revisions:** Adjustments to authority or location thresholds will be documented and incorporated as insights are gathered.

Trello Board:

https://trello.com/invite/b/66e2e5ff3822d226f9561e1a/ATTI53edf476ef9c23a885c5715a307d0e23A50FC622/gng1103-project-group-10