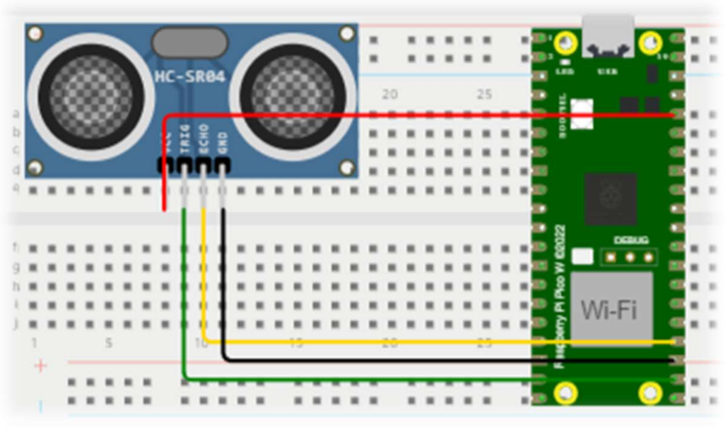
SMART PARKING

Development part 2:

Phase 4:

Raspberry pi module simulation:

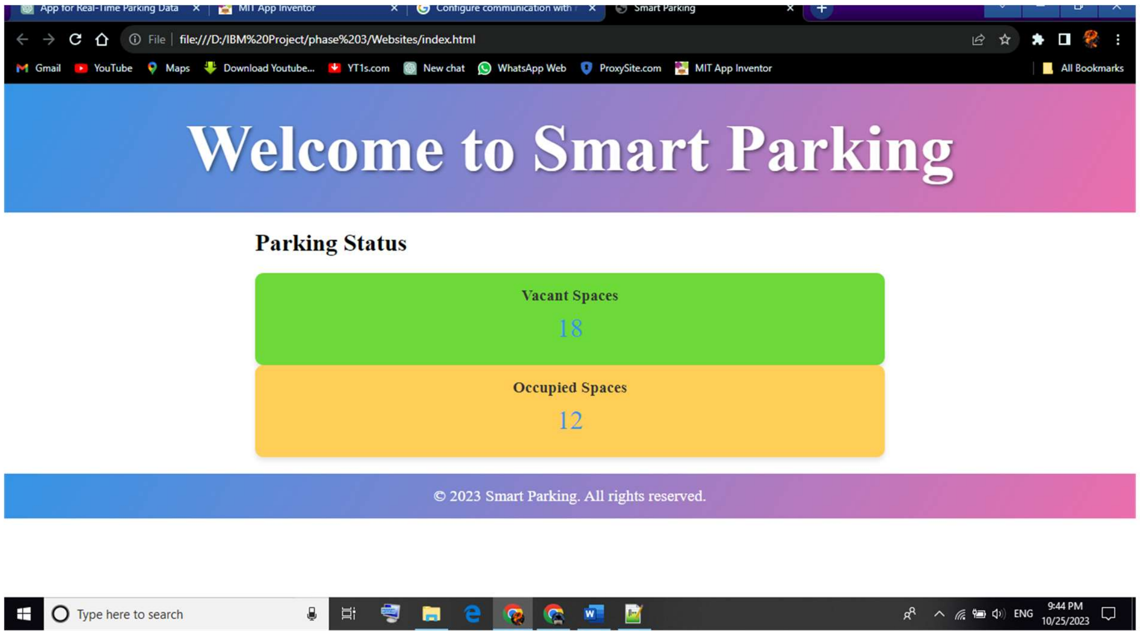


Python code:

```
import RPi.GPIO as GPIO

import time

import gspread

from oauth2client.service_account import ServiceAccountCredentials


# Set up Ultrasonic Sensor GPIO pins

TRIG_PIN = 17

ECHO_PIN = 18


# Initialize GPIO
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG_PIN, GPIO.OUT)
GPIO.setup(ECHO_PIN, GPIO.IN)

# Define Google Sheets-related variables
credentials_file = 'smart-parkingiot-095b74826f5a' # Replace with your credentials file
google_sheets_name = 'parkingdata' # Replace with your Google Sheets document name
sheet_name = 'Sheet1' # Replace with your sheet name

# Initialize Google Sheets
scope = ['https://www.googleapis.com/auth/spreadsheets']
credentials = ServiceAccountCredentials.from_json_keyfile_name(credentials_file, scope)
gc = gspread.authorize(credentials)
worksheet = gc.open(google_sheets_name).worksheet(sheet_name)

try:
  while True:
    # Measure distance using ultrasonic sensor
    GPIO.output(TRIG_PIN, False)
    time.sleep(2)

    GPIO.output(TRIG_PIN, True)
    time.sleep(0.00001)
    GPIO.output(TRIG_PIN, False)

    while GPIO.input(ECHO_PIN) == 0:
      pulse_start = time.time()

    while GPIO.input(ECHO_PIN) == 1:
      pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150  # Speed of sound (343 m/s)

    # Get current timestamp
    timestamp = time.strftime("%Y-%m-%d %H:%M:%S")

    # Determine occupancy status based on distance threshold
    distance_threshold = 50
    occupancy_status = "Occupied" if distance < distance_threshold else "Vacant"

    # Log data to Google Sheets
    worksheet.append_row([timestamp, distance, occupancy_status])

    print(f"Timestamp: {timestamp}, Distance: {distance} cm, Occupancy:
{occupancy_status}")
    time.sleep(5)  # Adjust the time interval as needed

except KeyboardInterrupt:
  GPIO.cleanup()
```

Website for smart parking:



Welcome to Smart Parking

Parking Status

Vacant Spaces
18

Occupied Spaces
12

© 2023 Smart Parking. All rights reserved.

This website is changed in mobile app

Set up your MIT App Inventor project

Visit the MIT App Inventor website (http://ai2.appinventor.mit.edu/).
Sign in with your Google account or create an account if you don't have one.
Click "Start New Project" to create a new project.

Design the user interface

Use App Inventor's visual designer to create user interfaces. You may want to include elements such as labels, buttons, and Web Viewer components. Design your interface to display parking data. For example, you can create a label or text field to display the number of available parking spaces.

Configure communication with Raspberry Pi

Use the App Inventor web component to communicate with the Raspberry Pi. Drag and drop web components from the palette to your screen.

In the web component properties, set the URL to your Raspberry Pi address, including the appropriate port and API endpoint to retrieve parking data.

Create buttons or user events that trigger web components to retrieve data from the Raspberry Pi. You can use the "Web.Get" method to send an HTTP GET request.

App first interface 1

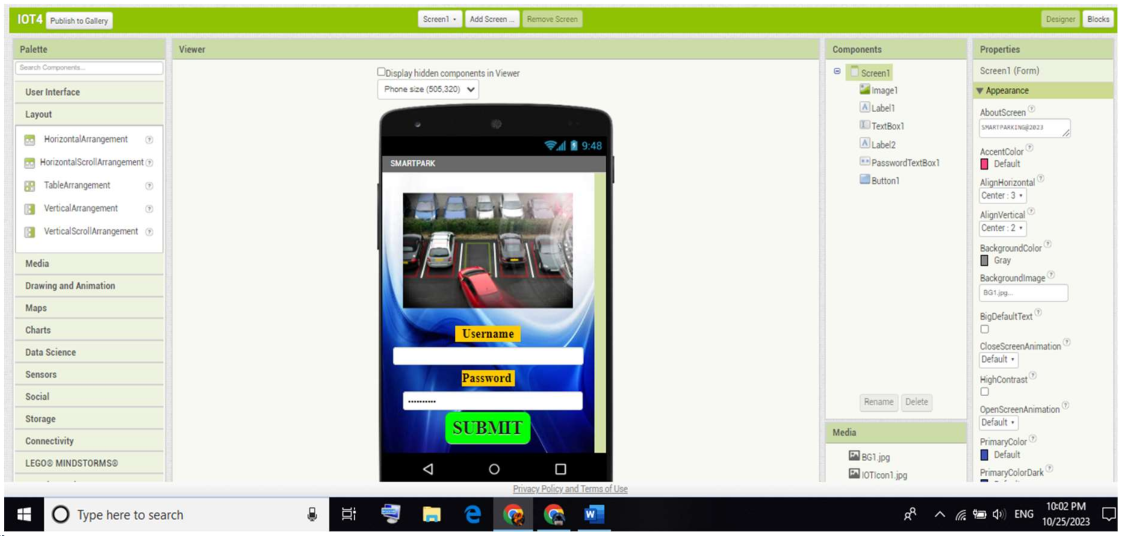


fig : screen1

App first interface 1

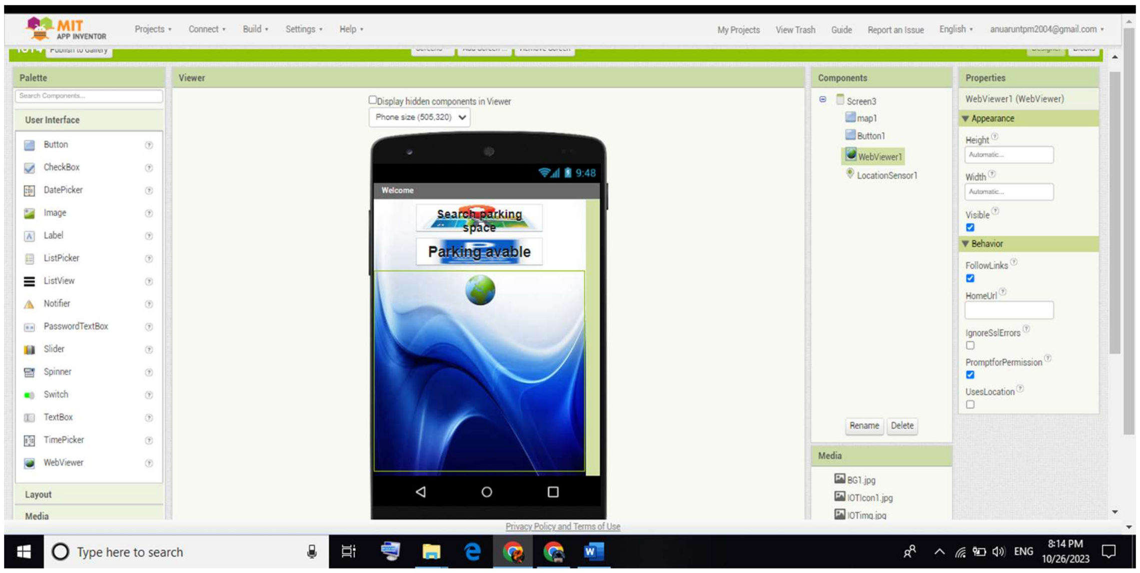
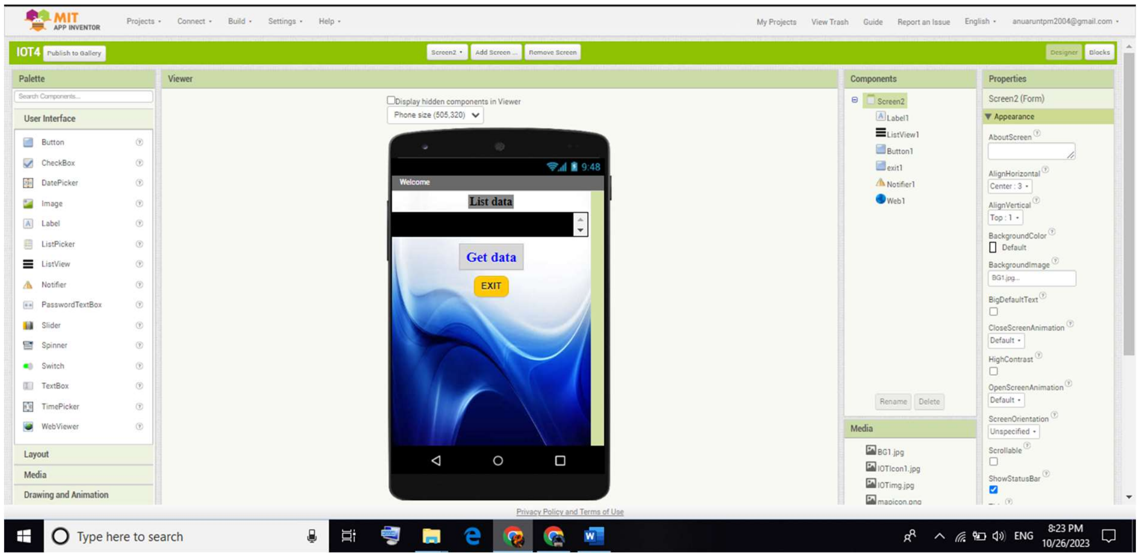


Fig: screen2

App first interface 1



AboutScreen
AlignHorizontal: Center : 3
AlignVertical: Top : 1
BackgroundColor: Default
BackgroundImage: BG1.jpg...
BigDefaultText: ☐
CloseScreenAnimation: Default
HighContrast: ☐
OpenScreenAnimation: Default
ScreenOrientation: Unspecified
Scrollable: ☐
ShowStatusBar: ✓

Receive and process data

Use the "Web.GotText" event to process data received from the Raspberry Pi. This event will be fired when a response is received.

In the "Web.GotText" event handler, you can analyze and process parking availability data, for example by updating user interface elements with received data .

View real-time data

Update your user interface elements (e.g. labels, text fields) with parking data obtained from the Raspberry Pi in the "Web.GotText" event handler.

Handle errors

Implement error handling in the "Web.GotText" event handler to resolve situations where the application may fail to connect to the Raspberry Pi or receive invalid data.

Test and debug

Use the MIT App Inventor emulator or a real device to test your app's functionality. Pay attention to how it communicates with the Raspberry Pi and displays data in real time.

Deploy

Once your app is ready, you can create an APK (Android Package) to distribute on Android devices. You can do this by clicking “Build” and selecting “App (provide QR code for .apk)”.

You can share apps via QR code, distribute via Google Play Store, or download to Android devices.

Continuous improvement and maintenance

Regularly update and maintain your app to improve performance and add new features as needed. Pay attention to user feedback and resolve any issues or feature requests.

Remember that this is a simplified overview and you may need to consider security, scalability, and advanced features, depending on your specific needs. Additionally, you should ensure that your Raspberry Pi is configured to transmit parking readiness data via HTTP or other suitable protocol.

User interactions and features

Improve the user experience by incorporating features such as a search function that allows users to search for parking spaces based on criteria such as location, price, and availability.

Implement functionality that allows users to reserve a parking space or get directions to a selected parking space.

You can create buttons to reserve a parking space or start navigation to the parking lot.

Real-time updates

To provide up-to-date parking availability data, consider setting up a mechanism in your app to periodically update data from your Raspberry Pi. Use a timer component to trigger data updates at specific intervals.

Geolocation and navigation

Integrate geolocation services to help users locate themselves and find nearby parking spaces. MIT App Inventor provides the "Location Sensor" component for this purpose.

Connect to mapping services like Google Maps and Map box to provide turn-by-turn navigation to your chosen parking space.

Analysis and user feedback

Collect user behaviour data using analytics tools like Firebase Analytics. This data helps us make informed decisions about improving the app.

Encourage users to provide feedback through the app. Create a link to a feedback form or user support email address.

Security and privacy

Ensure the security and privacy of user data and data transferred between your app and the Raspberry Pi. Implement authentication, encryption, and access control as required.

Comply with relevant data protection regulations and data protection standards, such as GDPR and CCPA, depending on the target group and location.

Scalability

Plan for scalability to accommodate a growing user base and increased parking space. Consider the architecture of your Raspberry Pi server so you can manage more clients if needed.

Regular updates and bug fixes

We regularly release app updates to fix bugs, security issues, and compatibility with the latest versions of mobile operating systems. We respond to user feedback and continually strive to improve the app in response to user needs and new challenges.

By following these steps and considering these factors, you can use MIT App Inventor to develop a robust and easy-to-use real-time parking availability mobile app. Successful app development requires continuous improvement and adaptation to meet user needs and overcome new challenges.

THANK YOU!