

# SMART PARKING

## PHASE 4: DEVELOPMENT PART 2

IoT Smart Parking project, it can enhance the functionality and user experience by incorporating web development technologies.

Here's how technologies into various aspects of the project:

### Web-based Dashboard for Administrators:

- Create a web-based dashboard for administrators to monitor and manage the parking system. This dashboard should provide real-time information about parking spot occupancy, reservations, and transaction history.
- Use web development technologies like HTML, CSS, and JavaScript, and consider using a web framework for efficiency.

**HTML/CSS:** Design the dashboard's layout and style using HTML and CSS.

**JavaScript:** Implement interactivity for real-time updates, charts, and user management.

**Web Framework:** You can use popular frameworks like React, Angular, or Vue.js for a organized and responsive interface.

### Mobile App:

- Develop a mobile app to reserve parking spots, make payments, and receive notifications.
- Use cross-platform mobile app development frameworks like React Native or Flutter to streamline app development for both Android and iOS.
- **API Integration:** Connect the app to the backend server for user authentication, reservation processing, and payment handling.

### Online Reservation System:

**React Native or Flutter:** Build the app's frontend using these frameworks, which allow you to write code once and deploy it on

Implement a web-based reservation system for students to check parking spot availability and make reservations.

- This system can be integrated with the mobile app and can be developed using standard web technologies.

**HTML/CSS:** Design the reservation interface.

**JavaScript:** Develop interactive features, such as selecting a parking spot and specifying the reservation duration.

**Backend:** Implement reservation logic on the server side, making use of frameworks like Express.js (Node.js) or Django (Python).

## Payment Gateway Integration:

If you include a payment system, you'll need to integrate a payment gateway into your web app for processing payment.

Popular payment gateways often provide APIs for this purpose. Here's a simplified example using Python and Flask:

**Flask:** Create an API endpoint to handle payment requests.

**Payment Gateway API:** Utilize the API provided by the payment gateway provider (e.g., Stripe, PayPal) for processing payments.

**Frontend Integration:** Integrate the payment process into your mobile app or web app, allowing users to enter payment details securely.

## Real-time Updates:

- Use web development technologies to ensure real-time updates on parking spot availability, reservation confirmation, and payment status.
- You can achieve this with technologies like WebSocket for real-time communication between the server and clients.

**WebSocket:** Implement WebSocket communication to push real-time updates to the web and mobile clients when a parking spot's status changes.

## User Authentication and Management:

For user authentication and management, you can create user registration and login systems within the mobile app and web interface.

Use web development technologies for user interfaces and backend logic:

**HTML/CSS:** Design registration and login forms.

**JavaScript:** Implement form validation and submission handling.

**Backend:** Create user accounts, manage authentication, and store user data securely in a database.

## Data Analytics and Reporting:

- Utilize web technologies to create data analytics and reporting features for administrators. You can use JavaScript libraries for data visualization and reporting tools.

**Data Visualization Libraries:** Integrate libraries like Chart.js or D3.js to display parking utilization statistics and trends.

**Backend:** Develop APIs for fetching historical parking data and generating reports.

## Python code:

```
Import time
```

```
data (0 for empty, 1 for occupied)
```

```
Parking_spots = [0, 0, 0, 0, 0]
```

```
Def get_parking_status():
```

```
Return [random.choice([0, 1]) for _ in range(len(parking_spots))]
```

```
While True:
```

```
Parking_spots = get_parking_status()
```

```
# Send parking_spots data to the cloud (simulated)
```

```
Print("Sending data to the cloud:", parking_spots) Time.sleep(10)
```

```
# Simulated data update every 10 seconds
```

## RASPBERRY PI INTEGRATION:

```
Import time
Import RPi.GPIO as GPIO
Import time
Import os,sys
From urllib.parse import urlparse
Import paho.mqtt.client as paho
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
Define pin for lcd
E_PULSE = 0.0005
E_DELAY = 0.0005
Delay = 1
LCD_RS = 7
LCD_E = 11
LCD_D4 = 12
LCD_D5 = 13
LCD_D6 = 15
LCD_D7 = 16
Slot1_Sensor = 29
Slot2_Sensor = 31
GPIO.setup(LCD_E, GPIO.OUT) # E
GPIO.setup(LCD_RS, GPIO.OUT) # RS
GPIO.setup(LCD_D4, GPIO.OUT) # DB4
GPIO.setup(LCD_D5, GPIO.OUT) # DB5
GPIO.setup(LCD_D6, GPIO.OUT) # DB6
GPIO.setup(LCD_D7, GPIO.OUT) # DB7
GPIO.setup(slot1_Sensor, GPIO.IN)
GPIO.setup(slot2_Sensor, GPIO.IN)
LCD_WIDTH = 16 # Maximum characters per line
LCD_CHR = True
LCD_CMD = False
LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
```

```

LCD_LINE_3 = 0x90# LCD RAM address for the 3rd line
Def on_connect(self, mosq, obj, rc):
Self.subscribe("Fan", 0)
Def on_publish(mosq, obj, mid):
Print("mid: " + str(mid))
Mqttc = paho.Client() declaration
Mqttc.on_connect = on_connect
Mqttc.on_publish = on_publish
url_str = os.environ.get('CLOUDMQTT_URL', 'tcp://broker.emqx.io:1883')
url = urlparse(url_str)
mqttc.connect(url.hostname, url.port)
Def lcd_init()
Lcd_byte(0x33,LCD_CMD) # 110011 Initialise
Lcd_byte(0x32,LCD_CMD) # 110010 Initialise
Lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
Lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
Lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
Lcd_byte(0x01,LCD_CMD) # 000001 Clear display
Time.sleep(E_DELAY)

```

## MIT APP INVENTOR:

- MIT App Inventor is a visual programming environment that allows you to create mobile applications for Android devices.
- You can use MIT App Inventor to develop a smart parking application that helps users find available parking spaces, reserve parking spots, and navigate to them.
- Here's a basic overview of how you can create a smart parking app using MIT App Inventor

## MOBILE APP:

Creating a mobile app for smart parking involves developing an application that helps users find and manage parking spaces efficiently. Here's a simplified guide on how to create a smart parking mobile app:

### 1. Define the Features:

- Start by outlining the features you want in your app. Common features include finding available parking spaces, reserving spots, navigation, and payment processing.

### 2. Choose a Development Platform:

- Decide on the technology stack for your app. Native development (iOS and Android), cross-platform development (using tools like Flutter or React Native), or web-based solutions (PWA) are some options.

### 3. Design the User Interface:

- Create wireframes and designs for your app's user interface. Ensure that it's user-friendly and easy to navigate.

### 4. Implement Parking Spot Data:

- Integrate a database or API to store and retrieve parking spot information. This may include location data, availability, pricing, and real-time updates.

### 5. Location Services:

- Use location services to determine the user's current location and display nearby parking spots on a map. You may need to request location permissions.

### 6. Real-time Updates:

- Implement real-time updates to display the current status of parking spots, whether they are available or occupied.

### 7. Reservation System:

- Create a reservation system that allows users to reserve parking spots in advance. This involves managing user accounts and payment processing.

### 8. Navigation:

- Integrate mapping and navigation services to help users find and navigate to their selected parking spot.

### 9. Payment Processing:

- Implement secure payment processing for parking reservations. You may need to integrate with payment gateways like Stripe or PayPal.

#### 10. User Profiles and Accounts:

- Allow users to create profiles, store payment information, and view their reservation history.

#### 11. Notifications:

- Send notifications to users about their reservations, parking availability, and other relevant information.

#### 12. Testing:

- Test your app thoroughly to ensure it works as expected. Test on various devices and simulate different scenarios.

#### 13. Security and Privacy:

- Implement security measures to protect user data and transactions. Ensure compliance with data privacy regulations.

#### 14. Deployment:

- Publish your app on the Google Play Store for Android devices and the Apple App Store for iOS devices.

#### 15. Marketing and User Engagement:

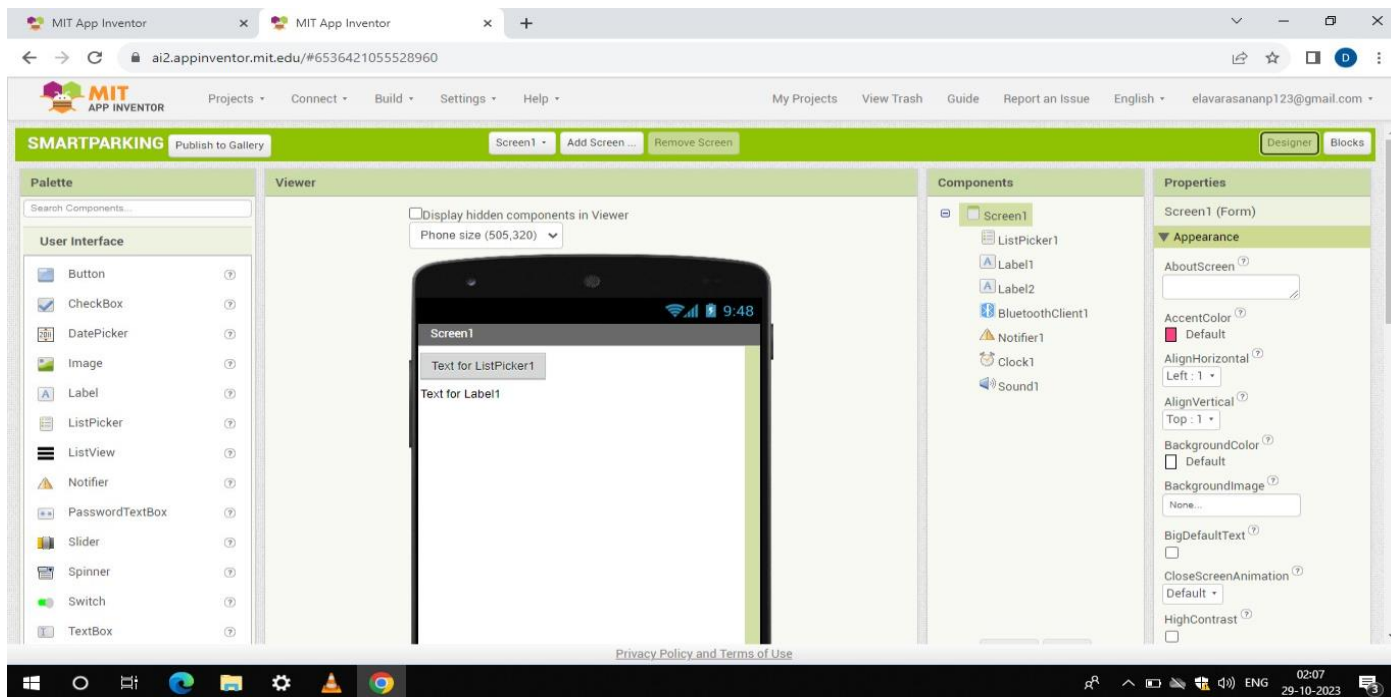
- Promote your app to attract users. Consider strategies like social media marketing, app store optimization, and partnerships with local businesses.

#### 16. User Support:

- Provide customer support through the app and offer assistance to users as needed.

- Creating a smart parking mobile app is a significant project, and it may require collaboration with developers, designers, and database administrators.

- Consider legal and regulatory aspects, such as data privacy and compliance with parking regulations in the areas your app will serve.



BLOCKS:



MIT App Inventor

ai2.appinventor.mit.edu/#6536421055528960

MIT APP INVENTOR

Projects Connect Build Settings Help

My Projects View Trash Guide Report an Issue

SMARTPARKING Publish to Gallery Screen1 Add Screen ... Remove Screen

Blocks

- Built-in
  - Control
  - Logic
  - Math
  - Text
  - Lists
  - Dictionaries
  - Colors
  - Variables
  - Procedures
- Screen1
  - ListPicker1
  - Label1
  - Label2
  - BluetoothClient1
  - Notifier1
  - Clock1
  - Sound1

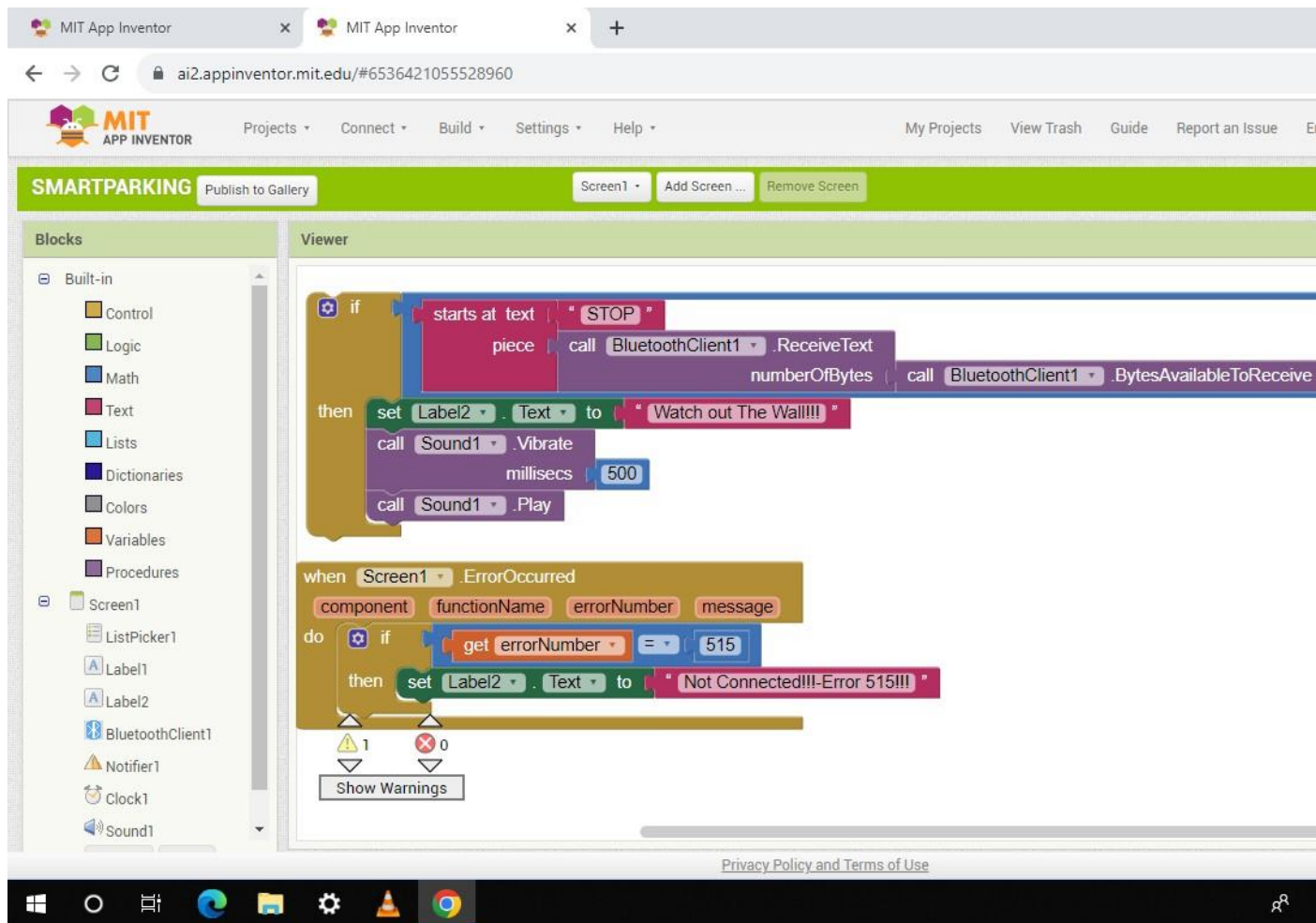
Viewer

```
when Screen1.Initialize
do
  if not BluetoothClient1.Enabled
  then
    call Notifier1.ShowAlert
      notice "Bluetooth is not enable-Use setting to turn BT on"
```

```
when ListPicker1.BeforePicking
do
  set ListPicker1.Elements to BluetoothClient1.AddressesAndNames
```

```
when ListPicker1.AfterPicking
do
  if call BluetoothClient1.Connect
    address ListPicker1.Selection
  Show Warnings ListPicker1.Elements to BluetoothClient1.AddressesAndNames
```

Privacy Policy and Terms of Use



THESE CODE AND IMAGES ARE INCLUDED IN  
PHASE 4:

AKASH.G

422621104004