# Development Guide for the Main System Using Django

**The main system includes the following modules:**

* ***Database Models:*** For storing data about owners, laptops, vehicles, and logs.
* ***Admin Interface:*** For managing and viewing registered laptops, vehicles, and logs.
* ***Laptop Registration Logic:*** Handles the registration of laptops.
* ***Authentication Logic:*** Validates the unique code, number plate, and passwords.
* ***Interfaces for Viewing and Tracking:*** Admin-facing pages for viewing and filtering data.

Step 1: Set Up the Django Project

1. Create a new Django project:

django-admin startproject main\_system

cd main\_system

1. Create an app for the system:

*python manage.py startapp gate\_pass*

1. Register the app in INSTALLED\_APPS in settings.py:

INSTALLED\_APPS = [

...

'gate\_pass',

]

Step 2: Database Models

***Define models for laptops, vehicles, owners, and logs in gate\_pass/models.py***.

from django.db import models

from django.utils.timezone import now

class Owner(models.Model):

first\_name = models.CharField(max\_length=50)

last\_name = models.CharField(max\_length=50)

email = models.EmailField(unique=True)

def \_\_str\_\_(self):

return f"{self.first\_name} {self.last\_name}"

class Laptop(models.Model):

serial\_number = models.CharField(max\_length=20, primary\_key=True)

mac\_address = models.CharField(max\_length=20)

model = models.CharField(max\_length=50)

unique\_code = models.CharField(max\_length=10, unique=True)

owner = models.ForeignKey(Owner, on\_delete=models.CASCADE)

def \_\_str\_\_(self):

return f"Laptop: {self.model} - {self.unique\_code}"

class Vehicle(models.Model):

number\_plate = models.CharField(max\_length=20, primary\_key=True)

owner = models.ForeignKey(Owner, on\_delete=models.CASCADE)

password = models.CharField(max\_length=255) # Password hashing applied during creation

def \_\_str\_\_(self):

return f"Vehicle: {self.number\_plate}"

class EntryExitLog(models.Model):

TYPE\_CHOICES = [

('Entry', 'Entry'),

('Exit', 'Exit'),

]

timestamp = models.DateTimeField(auto\_now\_add=True)

item\_type = models.CharField(max\_length=10, choices=[('Laptop', 'Laptop'), ('Vehicle', 'Vehicle')])

identifier = models.CharField(max\_length=50) # Unique code or number plate

status = models.CharField(max\_length=10, choices=TYPE\_CHOICES)

success = models.BooleanField()

def \_\_str\_\_(self):

return f"{self.item\_type} - {self.identifier} - {self.status}"

***Run migrations:***

python manage.py makemigrations

python manage.py migrate

Step 3: Admin Interface

***Register models in the admin site to allow viewing, adding, and filtering data***.

from django.contrib import admin

from .models import Owner, Laptop, Vehicle, EntryExitLog

@admin.register(Owner)

class OwnerAdmin(admin.ModelAdmin):

list\_display = ('first\_name', 'last\_name', 'email')

@admin.register(Laptop)

class LaptopAdmin(admin.ModelAdmin):

list\_display = ('serial\_number', 'model', 'unique\_code', 'owner')

search\_fields = ('serial\_number', 'unique\_code')

@admin.register(Vehicle)

class VehicleAdmin(admin.ModelAdmin):

list\_display = ('number\_plate', 'owner')

search\_fields = ('number\_plate',)

@admin.register(EntryExitLog)

class EntryExitLogAdmin(admin.ModelAdmin):

list\_display = ('item\_type', 'identifier', 'status', 'timestamp', 'success')

list\_filter = ('item\_type', 'status', 'success', 'timestamp')

Step 4: Laptop Registration Logic

***Define a view to handle laptop registration logic in gate\_pass/views.py.***

from django.http import JsonResponse

from django.views.decorators.csrf import csrf\_exempt

from .models import Laptop, Owner

import random, string

@csrf\_exempt

def register\_laptop(request):

if request.method == 'POST':

serial\_number = request.POST.get('serial\_number')

mac\_address = request.POST.get('mac\_address')

model = request.POST.get('model')

owner\_id = request.POST.get('owner\_id')

try:

owner = Owner.objects.get(id=owner\_id)

except Owner.DoesNotExist:

return JsonResponse({"message": "Owner not found"}, status=404)

unique\_code = ''.join(random.choices(string.ascii\_uppercase + string.digits, k=6))

Laptop.objects.create(

serial\_number=serial\_number,

mac\_address=mac\_address,

model=model,

unique\_code=unique\_code,

owner=owner

)

return JsonResponse({"message": "Laptop registered successfully", "unique\_code": unique\_code})

return JsonResponse({"message": "Invalid request"}, status=400)

Step 5: Authentication Logic

***Define views for authenticating laptops and vehicles.***

from django.views.decorators.csrf import csrf\_exempt

from .models import Laptop, Vehicle

@csrf\_exempt

def authenticate\_laptop(request):

if request.method == 'POST':

unique\_code = request.POST.get('unique\_code')

try:

laptop = Laptop.objects.get(unique\_code=unique\_code)

EntryExitLog.objects.create(identifier=unique\_code, item\_type="Laptop", status="Exit", success=True)

return JsonResponse({"message": "Laptop exit authenticated successfully"})

except Laptop.DoesNotExist:

return JsonResponse({"message": "Laptop not found"}, status=404)

@csrf\_exempt

def authenticate\_vehicle(request):

if request.method == 'POST':

number\_plate = request.POST.get('number\_plate')

password = request.POST.get('password')

try:

vehicle = Vehicle.objects.get(number\_plate=number\_plate)

if vehicle.password == password:

EntryExitLog.objects.create(identifier=number\_plate, item\_type="Vehicle", status="Exit", success=True)

return JsonResponse({"message": "Vehicle exit authenticated successfully"})

else:

return JsonResponse({"message": "Invalid password"}, status=400)

except Vehicle.DoesNotExist:

return JsonResponse({"message": "Vehicle not found"}, status=404)

*Step 6: URL Configuration*

***Add the endpoints to gate\_pass/urls.py.***

from django.urls import path

from . import views

urlpatterns = [

path('register-laptop/', views.register\_laptop, name='register\_laptop'),

path('authenticate-laptop/', views.authenticate\_laptop, name='authenticate\_laptop'),

path('authenticate-vehicle/', views.authenticate\_vehicle, name='authenticate\_vehicle'),

]

***Include these URLs in the project-level urls.py:***

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('api/', include('gate\_pass.urls')),

]

***Step 7: Viewing and Tracking Interface***

Leverage the Django Admin and custom templates for tracking vehicles and laptops.

***For custom pages, use Django’s ListView for filtering logs:***

from django.views.generic import ListView

from .models import EntryExitLog

class LogListView(ListView):

model = EntryExitLog

template\_name = 'gate\_pass/log\_list.html'

context\_object\_name = 'logs'

paginate\_by = 20

***Create a template gate\_pass/templates/gate\_pass/log\_list.html:***

{% extends "base\_generic.html" %}

{% block content %}

<h2>Entry and Exit Logs</h2>

<table>

<thead>

<tr>

<th>Item Type</th>

<th>Identifier</th>

<th>Status</th>

<th>Timestamp</th>

<th>Success</th>

</tr>

</thead>

<tbody>

{% for log in logs %}

<tr>

<td>{{ log.item\_type }}</td>

<td>{{ log.identifier }}</td>

<td>{{ log.status }}</td>

<td>{{ log.timestamp }}</td>

<td>{{ log.success }}</td>

</tr>

{% endfor %}

</tbody>

</table>

{% endblock %}

***Add this view to the URLs:***

path('logs/', LogListView.as\_view(), name='logs'),

# Development Guide for Kiosks (Portable Devices)

The kiosks serve as lightweight clients interacting with the main system. They provide interfaces for:

* **Entry**: Entering details of vehicles or laptops during entry.
* **Exit**: Validating details (unique codes, number plates, and passwords) during exit.
* **Sending Requests**: Communicating with the main server via HTTP requests.
* **Displaying Messages**: Showing success or error messages from the main system.

The implementation uses **C++** for speed and direct hardware interaction.

**Step 1: Development Environment Setup**

1. Install a C++ development environment:
   * Use **Visual Studio**, **Code::Blocks**, or **CLion** for development.
   * Ensure support for HTTP requests via a library like **cURL** or **Boost.Beast**.
2. Include dependencies for HTTP communication:
   * Install **cURL** library(bash)

sudo apt-get install libcurl4-openssl-dev

* + Link the library in your project.

**Step 2: Kiosk System Components**

**1. Entry Interface**

Allows security officers to enter vehicle number plates or laptop unique codes during entry.

**2. Exit Interface**

Prompts for number plates/unique codes and passwords for validation.

**3. Communication with the Main System**

Handles sending and receiving data from the main system.

**4. Display Messages**

Displays responses from the server.

**Step 3: Core Functionalities**

**Functionality: HTTP Request Handling**

Use the cURL library to send HTTP requests to the main system.

#include <iostream>

#include <string>

#include <curl/curl.h>

// Function to send HTTP POST request

std::string sendPostRequest(const std::string &url, const std::string &postData) {

CURL \*curl;

CURLcode res;

std::string response;

curl = curl\_easy\_init();

if (curl) {

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_POSTFIELDS, postData.c\_str());

// Write callback to capture server response

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, [](void \*ptr, size\_t size, size\_t nmemb, void \*stream) -> size\_t {

std::string \*response = static\_cast<std::string \*>(stream);

response->append(static\_cast<char \*>(ptr), size \* nmemb);

return size \* nmemb;

});

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

// Perform request

res = curl\_easy\_perform(curl);

if (res != CURLE\_OK) {

std::cerr << "cURL error: " << curl\_easy\_strerror(res) << std::endl;

}

curl\_easy\_cleanup(curl);

}

return response;

}

**Functionality: Entry Interface**

Accepts data for entry registration.

void handleEntry() {

std::string identifier, itemType;

std::cout << "Enter item type (Laptop/Vehicle): ";

std::cin >> itemType;

std::cout << "Enter identifier (Unique Code/Number Plate): ";

std::cin >> identifier;

// Prepare data

std::string postData = "item\_type=" + itemType + "&identifier=" + identifier;

// Send request

std::string url = "http://your-server-address/api/register-entry/";

std::string response = sendPostRequest(url, postData);

// Display server response

std::cout << "Server Response: " << response << std::endl;

}

**Functionality: Exit Interface**

Accepts identifier and password for validation.

void handleExit() {

std::string identifier, password, itemType;

std::cout << "Enter item type (Laptop/Vehicle): ";

std::cin >> itemType;

std::cout << "Enter identifier (Unique Code/Number Plate): ";

std::cin >> identifier;

std::cout << "Enter password: ";

std::cin >> password;

// Prepare data

std::string postData = "item\_type=" + itemType + "&identifier=" + identifier + "&password=" + password;

// Send request

std::string url = "http://your-server-address/api/register-exit/";

std::string response = sendPostRequest(url, postData);

// Display server response

std::cout << "Server Response: " << response << std::endl;

}

**Step 4: Kiosk Interface**

**Main Menu**

void displayMenu() {

int choice;

while (true) {

std::cout << "\nGate Pass System\n";

std::cout << "1. Entry\n";

std::cout << "2. Exit\n";

std::cout << "3. Exit Program\n";

std::cout << "Enter your choice: ";

std::cin >> choice;

switch (choice) {

case 1:

handleEntry();

break;

case 2:

handleExit();

break;

case 3:

std::cout << "Exiting program...\n";

return;

default:

std::cout << "Invalid choice. Please try again.\n";

}

}

}

**Step 5: Compilation and Execution**

1. Compile the program with the cURL library linked:

g++ -o kiosk kiosk.cpp -lcurl

1. ***Run the program(bash)***

./kiosk

**Step 6: Testing**

* **Entry Test**: Use sample data for number plates or unique codes and verify that the main system logs the entry.
* **Exit Test**: Input valid and invalid passwords to test server validation.

**Sample Server Responses**

* **Success**:

{"message": "Entry recorded successfully"}

* **Error**:

{"message": "Invalid password"}

**Step 7: Enhancements**

1. **User-Friendly Display**: Integrate with a hardware screen or terminal for better readability.
2. **Device Portability**: Install on portable hardware with built-in connectivity (e.g., Raspberry Pi or handheld devices).
3. **Real-Time Logging**: Display logs for successful entries and exits directly on the device.

# Integration and Deployment of Portable Devices into Hardware (ATM-Like Hardware)

This guide details how to load and integrate the kiosk system into portable ATM-like hardware, ensuring seamless communication with the main system deployed at https://sonye.pythonanywhere.com/gatepass.

**Step 1: Selecting the Best OS for Portable Devices**

**Recommended OS:**

* **Linux-based OS (e.g., Raspbian, Ubuntu Core):**
  + Lightweight, fast, and secure.
  + Widely used in embedded systems and portable devices.
  + Excellent support for hardware integration and communication protocols.

**Reasons:**

* Supports **C++ applications** out-of-the-box.
* Optimized for **resource-constrained environments**.
* Built-in support for networking (Wi-Fi, Ethernet, or Cellular).

**Step 2: Hardware Setup**

**Required Components:**

1. **Portable Device Hardware:**
   * A Raspberry Pi, handheld terminal, or industrial-grade portable device with:
     + Processor: ARM-based or equivalent.
     + Connectivity: Wi-Fi, Ethernet, or Cellular.
     + Input: Touchscreen or keypad.
     + Display: Small screen for message display.
2. **Peripheral Devices:**
   * Rechargeable battery for portability.
   * Built-in or external keyboard for input.

**Step 3: Installation Process**

**Step 3.1: Prepare the Device OS**

1. **Install Linux OS:**
   * For Raspberry Pi: Use **Raspberry Pi OS** (download from [raspberrypi.org](https://www.raspberrypi.org)).
   * Flash the OS onto an SD card using **balenaEtcher** or **Raspberry Pi Imager**.
2. **Boot and Configure Networking:**
   * Connect the device to a Wi-Fi network or configure cellular/Ethernet access.
   * Ensure the device can access the internet and communicate with external servers.
3. **Update OS Packages:((bash)**

sudo apt-get update && sudo apt-get upgrade -y

**Step 3.2: Install Dependencies for Kiosk Application**

1. Install the **cURL library** (for HTTP communication):

sudo apt-get install libcurl4-openssl-dev

1. Install a text editor (optional):

sudo apt-get install nano

**Step 3.3: Deploy the Kiosk Application**

1. Copy the compiled kiosk application (kiosk) to the device:
   * Use a USB drive or **scp** (secure copy):

scp kiosk user@device\_ip:/home/user/kiosk

1. Make the application executable:

chmod +x kiosk

**Step 4: Configuration to Communicate with the Main System**

**Step 4.1: Configure Server URL**

Edit the server URL in the kiosk application code to point to https://sonye.pythonanywhere.com/gatepass:

std::string url = "https://sonye.pythonanywhere.com/gatepass/api/";

Recompile the program if necessary:

g++ -o kiosk kiosk.cpp -lcurl

**Step 4.2: Configure Device Networking**

* Ensure the portable device is always connected to the internet.
* For Wi-Fi:
  + Edit the network configuration file (e.g., /etc/wpa\_supplicant/wpa\_supplicant.conf):

network={

ssid="YourNetworkSSID"

psk="YourNetworkPassword"

}

* + Restart the networking service:

sudo systemctl restart networking

**Step 4.3: Set Up Static IP (Optional)**

Assign a static IP to the device for easier identification and troubleshooting:

* Edit /etc/dhcpcd.conf:

interface wlan0

static ip\_address=192.168.1.100/24

static routers=192.168.1.1

static domain\_name\_servers=8.8.8.8

* + Restart the networking service.

**Step 5: Running the Kiosk Application**

1. Navigate to the directory containing the kiosk program:

cd /home/user/

1. Run the application:

./kiosk

1. **Testing Communication:**
   * Use the device to send test requests for entry and exit.
   * Verify responses from the main system.

**Step 6: Automation**

**Step 6.1: Auto-Start Kiosk Application on Boot**

1. Open the **rc.local** file:

sudo nano /etc/rc.local

1. Add the following line before exit 0:

/home/user/kiosk &

1. Save and exit.

**Step 6.2: Logging**

* Redirect logs to a file for debugging:

./kiosk > kiosk.log 2>&1 &

**Step 7: Hardware-Specific Enhancements**

**Step 7.1: Direct Interaction with Hardware**

* Use GPIO pins for direct hardware interaction (e.g., receipt printing or barcode scanning).
* Install the GPIO library:

sudo apt-get install wiringpi

* Example for toggling GPIO pins:

#include <wiringPi.h>

int main() {

wiringPiSetup();

pinMode(0, OUTPUT);

digitalWrite(0, HIGH);

delay(1000);

digitalWrite(0, LOW);

return 0;

}

**Step 7.2: Portable Power**

* Use a rechargeable battery pack designed for Raspberry Pi or similar devices.

**Step 8: Monitoring and Maintenance**

1. **Remote Access:**
   * Install **SSH** to remotely manage the device:

sudo apt-get install openssh-server

* + Enable SSH:

sudo systemctl enable ssh

sudo systemctl start ssh

1. **Regular Updates:**
   * Update OS and application regularly:

sudo apt-get update && sudo apt-get upgrade -y

1. **Error Handling:**
   * Log errors and response statuses from the main system to identify issues.