INTRODUCTION

PROBLEM STATEMENT: Due to the increased number of vehicles, parking in public places becomes an issue. There is no guaranteed space for parking while visiting public spaces. This creates a lot of havoc amongst people. Parking Management is done manually which is not very efficient which makes it difficult to keep track of the available parking spaces in the area. This project addresses this issue.

SOLUTION: Hence, it is required to make a Parking Management System in order to reserve parking spaces in advance. This makes this process efficient and easy. If data is stored in a systematic format, arranged in tabular columns, with information of which parking slots are free and which are occupied, it makes it easier to manage the data and thus it is less prone to errors. Searching through manual records is a time-consuming work which must be avoided. Creating a parking management system that stores all its data in a compact form in a database makes this task much easier.

ADVANTAGES:

➢ Versatile: It is very easy for authorities and vehicle owners to use it. It is also very flexible and does not cause any inconvenience to its users.

➢ Easy to manage: Because parking management systems are organized in a structured manner, it is very easy to manage as well as control and regulate data.

➢ Reservation: Allows users to register and reserve parking spaces for a particular time slot in advance

➢ Easy access to data: All the entries of all users on different days can be easily accessed in a compact database, with details of each of the registered users.

OBJECTIVE

- To enable drivers to easily locate and reserve a parking spot online through our web platform.
- To reduce traffic and make parking hassle-free.
- To make fee payment easier and quicker.

SCOPE

- Congestion of vehicles due to previous manual method can be limited.
- Can easily be used in public parking spaces.

MODULES

- Streamlit: Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps. It is a Python-based library specifically designed for machine learning engineers. It's a tool that is easy to learn and use, and can display data and collect needed parameters for modelling on the most basic level.
- Base64: base64 is a binary to text encoding scheme that represents binary data in an American Standard Code for Information Interchange (ASCII) string format. It's designed to carry data stored in binary format across the channels, and it takes any form of data and transforms it into a long string of plain text.
- PIL: Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.
- Pandas: It is a Python library used for working with data sets. It has functions for analysing, cleaning, exploring, and manipulating data. It is made mainly for working with relational or labelled data.
- Streamlit_lottie: This module aims for specific animation elements and parameters, and it allows you to add interactivity and change parameters during runtime.

SYSTEM IMPLEMENTATION

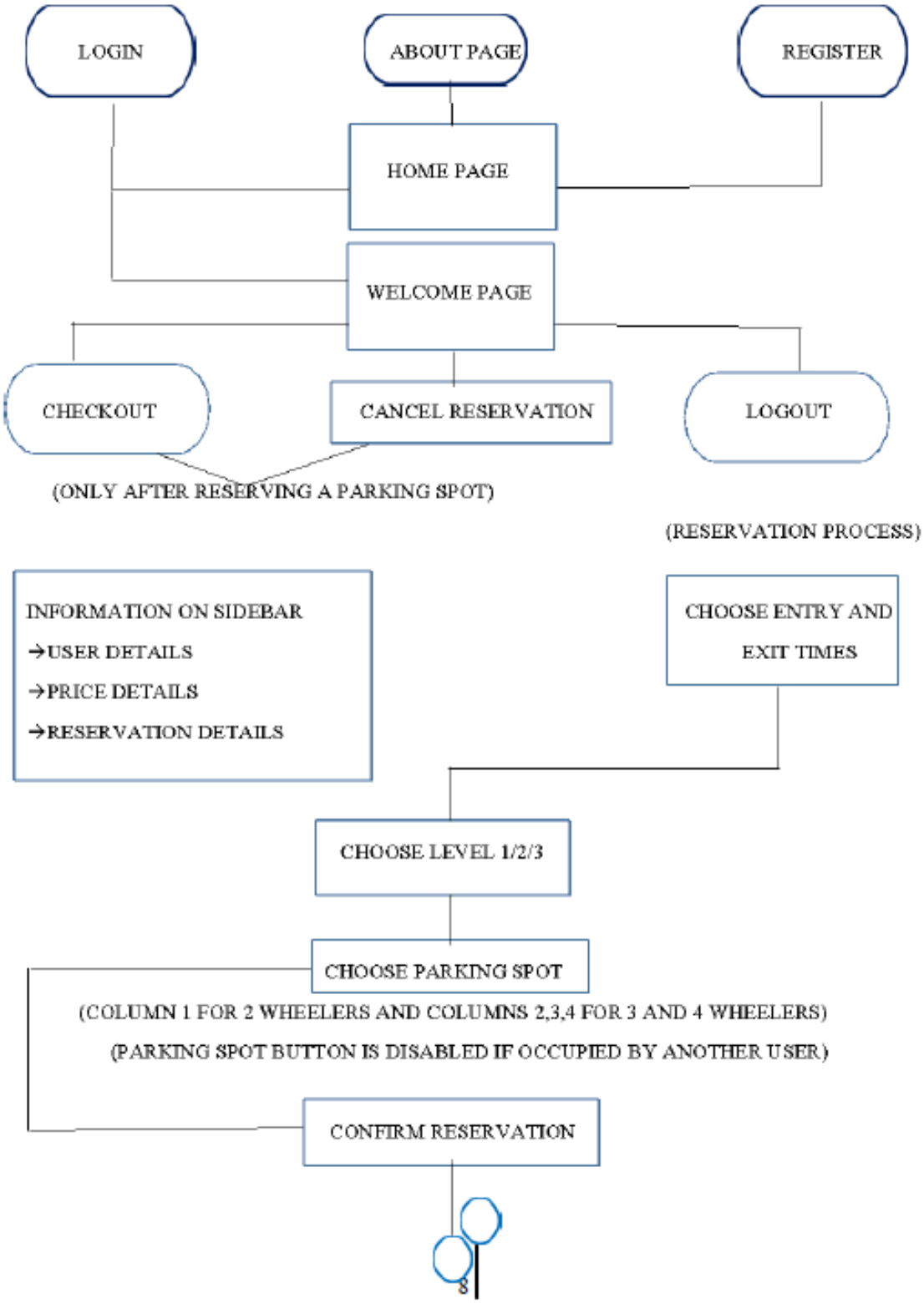
SOFTWARE REQUIREMENTS:

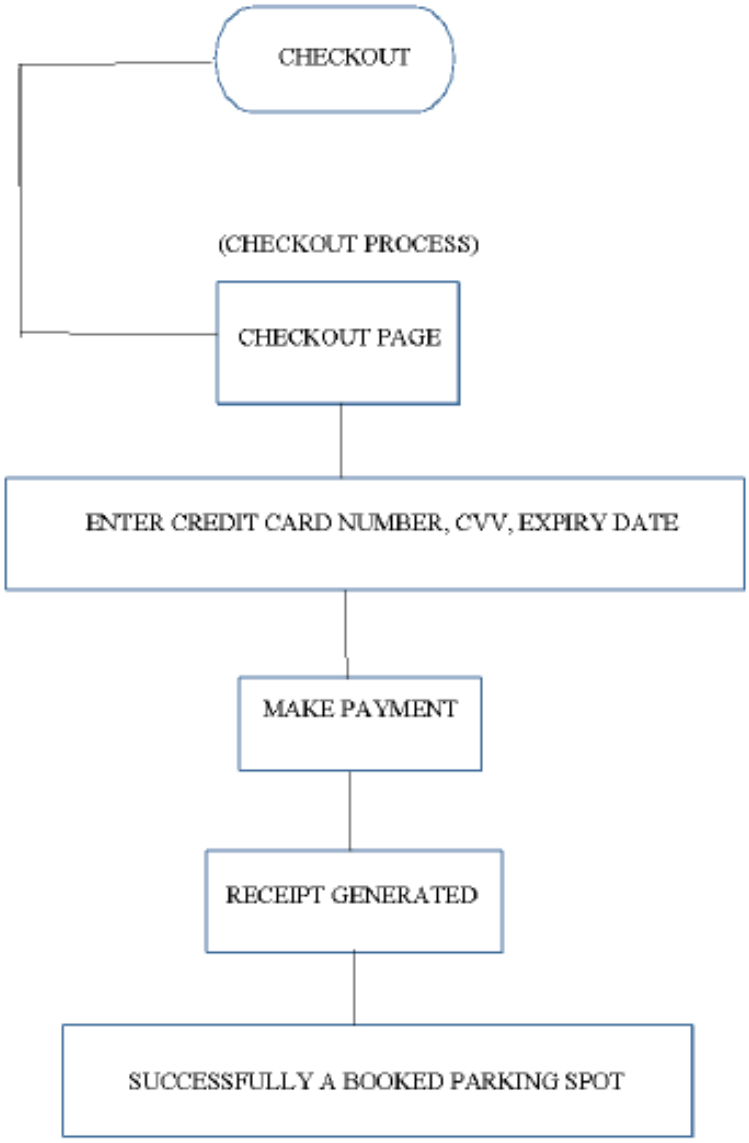
- Windows 10/11
- web browser
- Python 3.8 to 3.11 installed on Windows
- mysql server
- Python Libraries:
- streamlit
- mysql.connector
- pandas for tables
- PIL Image library
- streamlit_lottie
- base64

HARDWARE REQUIREMENTS:

- Intel Core i3 CPU or above
- 4 GB of RAM
- 30GB Free space

DATA FLOW DIAGRAM




CHECKOUT
(CHECKOUT PROCESS)
CHECKOUT PAGE
ENTER CREDIT CARD NUMBER, CVV, EXPIRY DATE
MAKE PAYMENT
RECEIPT GENERATED
SUCCESSFULLY A BOOKED PARKING SPOT

CODE WINDOW

PYTHON CODE:

```python
#Import necessary libraries
import os
import io
import math
import bcrypt
import base64
import mysql.connector as mysql
from datetime import datetime, date, timedelta
from PIL import Image, ImageDraw, ImageFont

import streamlit as st
import pandas as pd

# ==================== CONFIG ====================
# Database config - update to your MySQL settings
DB_CONFIG = {
    "host": "localhost",
    "user": "root",
    "password": "12345678",
    "database": "parking"
}

# Image paths (change if needed). Use relative paths or URLs.
BACKGROUND_IMAGE_PATH = "D:\\Skills Dev\\Python project\\Background.jpg"   # used on home page
CAR_IMAGE_PATH = "D:\\Skills Dev\\Python project\\kindpng_76524.png"               # shown on level selection or reservation

# Parking configuration
SLOTS_PER_LEVEL = 20
LEVELS = [1, 2, 3]

# Rates (per hour)
RATES = {
    "2 wheeler": 10.0,
    "4 wheeler": 20.0,
    "3 wheeler": 12.5
}
```

```python

# ==================== UTILITIES ====================
def get_db_conn():
    return mysql.connect(
        host="localhost",
        user="root",
        password="12345678",
        database="parking",
        autocommit=True
    )
def init_db():
    """Create required tables if they don't exist."""
    conn = get_db_conn()
    cur = conn.cursor()
    # user_details table
    cur.execute("""
    CREATE TABLE IF NOT EXISTS user_details (
        user_id VARCHAR(100) PRIMARY KEY,
        user_name VARCHAR(255),
        user_password VARBINARY(60),
        user_addr VARCHAR(500),
        vehicle_no VARCHAR(50),
        user_mobile_no VARCHAR(20),
        vehicle_type VARCHAR(20),
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    )
    """)
```

```python
    # reservations table - one central table (better than per-user tables)
    cur.execute("""
    CREATE TABLE IF NOT EXISTS reservations (
        reservation_id INT AUTO_INCREMENT PRIMARY KEY,
        user_id VARCHAR(100),
        level_no INT,
        slot_no INT,
        entry_datetime DATETIME,
        exit_datetime DATETIME,
        vehicle_type VARCHAR(20),
        status VARCHAR(20) DEFAULT 'reserved', -- reserved, cancelled, paid, completed
        bill_amount DOUBLE DEFAULT 0,
        paid TINYINT(1) DEFAULT 0,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (user_id) REFERENCES user_details(user_id)
    )
    """)
    cur.close()
    conn.close()

def hash_password(plain_password: str) -> bytes:
    """Return bcrypt hashed password (bytes)."""
    return bcrypt.hashpw(plain_password.encode("utf-8"), bcrypt.gensalt())

def check_password(plain_password, hashed):
    # hashed must be bytes
    return bcrypt.checkpw(plain_password.encode("utf-8"), hashed)

```

```python

def image_to_base64(path):
    if not os.path.exists(path):
        return None
    with open(path, "rb") as f:
        return base64.b64encode(f.read()).decode()

# ==================== DB ACTIONS ====================
def register_user(user_id, name, password, addr, vehicle_no, mobile, vehicle_type):
    conn = get_db_conn()
    cur = conn.cursor()
    try:
        # Hash password (bytes)
        hashed_pw = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
        cur.execute("""
            INSERT INTO user_details (user_id,user_name,user_password,user_addr,vehicle_no,user_mobile_no,vehicle_type)
            VALUES (%s,%s,%s,%s,%s,%s,%s)
        """, (user_id, name, hashed_pw, addr, vehicle_no, mobile, vehicle_type))
        conn.commit()
        return True, None
    except mysql.Error as e:
        return False, str(e)
    finally:
        cur.close()
        conn.close()
```

```python

def get_user(user_id):
    conn = get_db_conn()
    cur = conn.cursor()
    cur.execute("""
        SELECT user_id,user_name,user_password,user_addr,vehicle_no,user_mobile_no,vehicle_type
        FROM user_details WHERE user_id=%s
    """, (user_id,))
    row = cur.fetchone()
    cur.close()
    conn.close()
    if not row:
        return None
    # Ensure password is bytes
    password_hash = row[2]
    if isinstance(password_hash, str):
        password_hash = password_hash.encode("utf-8")  # convert to bytes
    return {
        "user_id": row[0],
        "name": row[1],
        "password_hash": password_hash,  # now definitely bytes
        "address": row[3],
        "vehicle_no": row[4],
        "mobile_no": row[5],
        "vehicle_type": row[6]
    }
```

```python

def user_exists(user_id):
    return get_user(user_id) is not None

def authenticate_user(user_id, password):
    user = get_user(user_id)
    if not user:
        return False
    return check_password(password, user["password_hash"])

def create_reservation_db(user_id, level_no, slot_no, entry_dt, exit_dt, vehicle_type, bill_amount=0.0):
    conn = get_db_conn()
    cur = conn.cursor()
    cur.execute("""
        INSERT INTO reservations (user_id, level_no, slot_no, entry_datetime, exit_datetime, vehicle_type, bill_amount, status, paid)
        VALUES (%s,%s,%s,%s,%s,%s,%s,%s,'reserved',0)
    """, (user_id, level_no, slot_no, entry_dt, exit_dt, vehicle_type, bill_amount))
    conn.commit()
    cur.close()
    conn.close()

def reservations_for_user(user_id):
    conn = get_db_conn()
    df = pd.read_sql("SELECT reservation_id, level_no, slot_no, entry_datetime, exit_datetime, vehicle_type, status, bill_amount, paid FROM reservations WHERE user_id=%s ORDER BY created_at DESC",
                     conn, params=(user_id,))
    conn.close()
    return df
```

```python

def get_overlapping_reserved_slots(level_no, entry_dt, exit_dt):
    """Return set of slot numbers in `level_no` that overlap with given period and not cancelled."""
    conn = get_db_conn()
    cur = conn.cursor()
    cur.execute("""
        SELECT slot_no FROM reservations
        WHERE level_no=%s AND status IN ('reserved','paid') AND NOT (exit_datetime <= %s OR entry_datetime >= %s)
    """, (level_no, entry_dt, exit_dt))
    rows = cur.fetchall()
    cur.close()
    conn.close()
    return set(r[0] for r in rows)

def compute_cost(vehicle_type, entry_dt, exit_dt):
    seconds = (exit_dt - entry_dt).total_seconds()
    hours = math.ceil(seconds / 3600)
    key = str(vehicle_type).lower()
    if "2" in key:
        rate = RATES["2 wheeler"]
    elif "3" in key:
        rate = RATES["3 wheeler"]
    else:
        rate = RATES["4 wheeler"]
    return rate * max(1, hours), hours
```

```python

def mark_reservation_paid(reservation_id, amount):
    conn = get_db_conn()
    cur = conn.cursor()
    cur.execute("UPDATE reservations SET paid=1, bill_amount=%s, status='paid' WHERE reservation_id=%s", (amount, reservation_id))
    conn.commit()
    cur.close()
    conn.close()

def cancel_reservation_db(reservation_id, user_id):
    """Cancel only if the reservation belongs to the user and entry_datetime is in future and status reserved."""
    conn = get_db_conn()
    cur = conn.cursor()
    # verify conditions
    cur.execute("SELECT entry_datetime, status FROM reservations WHERE reservation_id=%s AND user_id=%s", (reservation_id, user_id))
    row = cur.fetchone()
    if not row:
        cur.close()
        conn.close()
        return False, "Reservation not found."
    entry_dt, status = row[0], row[1]
    if status != "reserved":
        cur.close()
        conn.close()
        return False, "Only reserved reservations can be cancelled."
    if entry_dt <= datetime.now():
        cur.close()
        conn.close()
        return False, "Cannot cancel a reservation whose entry time has passed or is ongoing."
    cur.execute("UPDATE reservations SET status='cancelled' WHERE reservation_id=%s", (reservation_id,))
    conn.commit()
    cur.close()
    conn.close()
    return True, None

```

```python

# ================== UI - Helpers ==================
def set_background_image():
    b64 = image_to_base64(BACKGROUND_IMAGE_PATH)
    if not b64:
        return
    page_bg_img = f"""
    <style>
    .stApp {{
      background-image: url("{b64}");
      background-size: cover;
      background-position: center;
    }}
    .center-box {{
      display:flex;
      align-items:center;
      justify-content:center;
      height:60vh;
      flex-direction:column;
      text-align:center;
      color: white;
      text-shadow: 1px 1px 3px #000;
    }}
    </style>
    """
    st.markdown(page_bg_img, unsafe_allow_html=True)

def make_receipt_png(receipt_info: dict) -> io.BytesIO:
    """Create a PNG receipt (PIL) and return BytesIO."""
    width, height = 800, 500
    img = Image.new("RGB", (width, height), color="white")
    draw = ImageDraw.Draw(img)
    try:
        font = ImageFont.truetype("arial.ttf", 20)
    except Exception:
        font = ImageFont.load_default()

```

```python

    y = 30
    draw.text((40, y), "VPark Receipt", font=font)
    y += 40
    for k, v in receipt_info.items():
        draw.text((40, y), f"{k}: {v}", font=font)
        y += 30

    # small footer
    draw.text((40, y+20), "Thank you for using VPark!", font=font)
    buf = io.BytesIO()
    img.save(buf, format="PNG")
    buf.seek(0)
    return buf

# ================== PAGES ==================
def home_page():
    st.set_page_config(page_title="VPark", layout="wide", initial_sidebar_state="collapsed")
    set_background_image()

    # Top-right About button
    cols = st.columns([8,1])
    with cols[1]:
        if st.button("About Us"):
            st.session_state.page = "about"

```

```python

    st.markdown("<div class='center-box'>", unsafe_allow_html=True)
    st.markdown("<h1 style='font-size:100px;text-align:center;margin:2;color:white;'>VPark</h1>", unsafe_allow_html=True)
    st.write("")
    c1, c2, c3 = st.columns([3,1,3])
    with c2:
        if st.button("Login", key="home_login"):
            st.session_state.page = "login"
        st.write("")
        if st.button("Sign Up", key="home_signup"):
            st.session_state.page = "signup"
    st.markdown("</div>", unsafe_allow_html=True)

```

```python

def about_page():
    st.header("About VPark")
    st.write("Smart Parking system built with Streamlit and MySQL.")
    st.write("- Signup/login, reservations, billing, receipt download.")
    st.write("- Demo payment (simulated) — do not use for real card entry.")
    if st.button("Back"):
        st.session_state.page = "home"

# ---------- Signup ----------
def signup_page():
    st.header("Sign Up")
    with st.form("signup_form"):
        uid = st.text_input("User ID")
        name = st.text_input("Full Name")
        pw = st.text_input("Password", type="password")
        addr = st.text_area("Address")
        vehicle_no = st.text_input("Vehicle Number")
        mobile = st.text_input("Mobile Number")
        vehicle_type = st.selectbox("Vehicle Type", ["2 wheeler","3 wheeler","4 wheeler"])
        submitted = st.form_submit_button("Create Account")
        if submitted:
            if user_exists(uid):
                st.error("User ID already exists")
            else:
                ok, err = register_user(uid, name, pw, addr, vehicle_no, mobile, vehicle_type)
                if ok:
                    st.success("Account created! Please login.")
                    st.session_state.page = "login"
                else:
                    st.error(f"Error: {err}")
    if st.button("Back to Home"):
        st.session_state.page = "home"
```

```python
# ---------- Login ----------
def login_page():
    st.header("Login")
    with st.form("login_form"):
        uid = st.text_input("User ID")
        pw = st.text_input("Password", type="password")
        submitted = st.form_submit_button("Login")
        if submitted:
            if authenticate_user(uid, pw):
                st.success("Logged in!")
                st.session_state.user_id = uid
                st.session_state.page = "welcome"
            else:
                st.error("Invalid credentials")
    if st.button("Back to Home"):
        st.session_state.page = "home"

# ---------- Sidebar user info (used on many pages) ----------
def show_sidebar_user_info():
    if "user_id" not in st.session_state:
        return
    user = get_user(st.session_state.user_id)
    if not user:
        return
    st.sidebar.title("Account")
    st.sidebar.markdown(f"**{user['name']}**")
    st.sidebar.markdown(f"Vehicle: {user['vehicle_no']}")
    st.sidebar.markdown(f"Mobile: {user['mobile_no']}")
    st.sidebar.markdown(f"Type: {user['vehicle_type']}")
    st.sidebar.markdown("---")

```

```python
    # Pending bills
    df = reservations_for_user(user["user_id"])
    pending = df[(df["paid"] == 0) & (df["status"] == "reserved")]
    if not pending.empty:
        st.sidebar.markdown("### Pending Bills")
        for _, row in pending.iterrows():
            rid = int(row["reservation_id"])
            entry = pd.to_datetime(row["entry_datetime"])
            exit_ = pd.to_datetime(row["exit_datetime"])
            amt, hours = compute_cost(row["vehicle_type"], entry, exit_)
            st.sidebar.markdown(f"- Res {rid}: {entry.strftime('%Y-%m-%d %H:%M')} → {exit_.strftime('%Y-%m-%d %H:%M')} → {amt:.2f}")
            if st.sidebar.button(f"Pay {rid}", key=f"pay_sidebar_{rid}"):
                st.session_state.selected_reservation = rid
                st.session_state.page = "bill"
    st.sidebar.markdown("---")
    if st.sidebar.button("Logout"):
        st.session_state.clear()
        st.session_state.page = "home"
        st.experimental_rerun()

```

```python

# ---------- Welcome ----------
def welcome_page():
    set_background_image()

    if "user_id" not in st.session_state:
        st.info("Please login.")
        st.session_state.page = "login"
        return
    show_sidebar_user_info()
    user = get_user(st.session_state.user_id)
    st.markdown(f"<h2 style='text-align:center;'>Welcome {user['name']}!</h2>", unsafe_allow_html=True)
    c1, c2, c3 = st.columns(3)
    with c1:
        if st.button("Account History"):
            st.session_state.page = "history"
    with c2:
        if st.button("Reservation"):
            # initialize reservation flow
            st.session_state.pop("reservation", None)
            st.session_state.page = "reserve_time"
    with c3:
        if st.button("Current Bills / Checkout"):
            st.session_state.page = "bill"

```

```python
# ---------- Reservation - select time ----------
def reserve_time_page():
    if "user_id" not in st.session_state:
        st.session_state.page = "login"
        return
    show_sidebar_user_info()
    st.header("Reservation - Select Entry & Exit")
    user = get_user(st.session_state.user_id)
    with st.form("time_form"):
        now = datetime.now()
        entry_date = st.date_input("Entry Date", value=now.date(), min_value=now.date())
        entry_time = st.time_input("Entry Time", value=now.time().replace(second=0, microsecond=0))
        exit_date = st.date_input("Exit Date", value=(now + timedelta(hours=1)).date(), min_value=entry_date)
        exit_time = st.time_input("Exit Time", value=(now + timedelta(hours=1)).time().replace(second=0, microsecond=0))
        submitted = st.form_submit_button("Next: Choose Level")
        if submitted:
            entry_dt = datetime.combine(entry_date, entry_time)
            exit_dt = datetime.combine(exit_date, exit_time)
            if exit_dt <= entry_dt:
                st.error("Exit must be after entry time.")
            else:
                st.session_state.reservation = {
                    "entry_dt": entry_dt,
                    "exit_dt": exit_dt,
                    "vehicle_type": user["vehicle_type"]
                }
                st.session_state.page = "choose_level"
    if st.button("Back"):
        st.session_state.page = "welcome"
```

```python
# ---------- Reservation - choose level ----------
def choose_level_page():
    show_sidebar_user_info()
    st.header("Choose Parking Level")
    cols = st.columns(len(LEVELS))
    if os.path.exists(CAR_IMAGE_PATH):
        st.image(CAR_IMAGE_PATH, width=200)
    for i, level in enumerate(LEVELS):
        with cols[i]:
            if st.button(f"Level {level}"):
                st.session_state.reservation["level"] = level
                st.session_state.page = "choose_slot"
    if st.button("Back"):
        st.session_state.page = "reserve_time"
```

```python
# ---------- Reservation - choose slot ----------
def choose_slot_page():
    show_sidebar_user_info()
    res = st.session_state.get("reservation")
    if not res:
        st.session_state.page = "reserve_time"
        return
    entry_dt = res["entry_dt"]
    exit_dt = res["exit_dt"]
    level = res["level"]
    st.header(f"Choose Slot — Level {level}")
    st.write(f"{entry_dt.strftime('%Y-%m-%d %H:%M')} → {exit_dt.strftime('%Y-%m-%d %H:%M')}")
    reserved = get_overlapping_reserved_slots(level, entry_dt, exit_dt)

    cols_per_row = 5
    total = SLOTS_PER_LEVEL
    for i in range(0, total, cols_per_row):
        cols = st.columns(cols_per_row)
        for j, col in enumerate(cols):
            slot_no = i + j + 1
            key = f"slot_{level}_{slot_no}"
            if slot_no in reserved:
                # dimmed - show disabled button
                col.markdown(f"<button disabled style='opacity:0.5;padding:8px 12px'>Slot {slot_no} (Used)</button>", unsafe_allow_html=True)
            else:
                if col.button(f"Slot {slot_no}", key=key):
                    st.session_state.reservation["slot_no"] = slot_no
                    st.session_state.page = "confirm_reservation"
                    st.stop()
    if st.button("Back"):
        st.session_state.page = "choose_level"
```

```python
# ---------- Confirm reservation ----------
def confirm_reservation_page():
    show_sidebar_user_info()
    r = st.session_state.get("reservation")
    if not r:
        st.session_state.page = "reserve_time"
        return
    st.header("Confirm Reservation")
    st.write(f"- User: {st.session_state.user_id}")
    st.write(f"- Level: {r['level']}")
    st.write(f"- Slot: {r['slot_no']}")
    st.write(f"- Entry: {r['entry_dt'].strftime('%Y-%m-%d %H:%M')}")
    st.write(f"- Exit: {r['exit_dt'].strftime('%Y-%m-%d %H:%M')}")
    if st.button("Confirm Reservation"):
        # compute bill but keep unpaid
        amount, hours = compute_cost(r["vehicle_type"], r["entry_dt"], r["exit_dt"])
        create_reservation_db(st.session_state.user_id, r["level"], r["slot_no"], r["entry_dt"], r["exit_dt"], r["vehicle_type"], amount)
        st.success("Reservation created.")
        st.session_state.page = "welcome"
    if st.button("Back"):
        st.session_state.page = "choose_slot"
```

```python
# ---------- Billing page ----------
def bill_page():
    show_sidebar_user_info()
    user_id = st.session_state.get("user_id")
    df = reservations_for_user(user_id)
    # If user selected a reservation via sidebar, use it
    selected = st.session_state.get("selected_reservation")
    if selected:
        row = df[df["reservation_id"] == int(selected)]
        if row.empty:
            st.error("Selected reservation not found.")
            return
        row = row.iloc[0]
    else:
        pending = df[(df["paid"] == 0) & (df["status"] == "reserved")]
        if pending.empty:
            st.info("No pending bills. Create a reservation first.")
            if st.button("Back to Welcome"):
                st.session_state.page = "welcome"
            return
        row = pending.iloc[0]

    st.header("Bill & Checkout")
    st.write(f"Reservation ID: {row['reservation_id']}")
    entry = pd.to_datetime(row["entry_datetime"])
    exit_ = pd.to_datetime(row["exit_datetime"])
    amount, hours = compute_cost(row["vehicle_type"], entry, exit_)
    st.write(f"Vehicle: {row['vehicle_type']}")
    st.write(f"Duration (rounded): {hours} hours")
    st.write(f"Amount Due: {amount:.2f}")
    col1, col2 = st.columns([1,1])
    with col1:
        if st.button("Proceed to Payment"):
            st.session_state.current_bill = {"reservation_id": int(row["reservation_id"]), "amount": float(amount)}
            st.session_state.page = "payment"
    with col2:
        if st.button("Back"):
            st.session_state.page = "welcome"

# ---------- Payment page (simulated) ----------
def payment_page():
    show_sidebar_user_info()
    bill = st.session_state.get("current_bill")
    if not bill:
        st.error("No bill selected.")
        st.session_state.page = "welcome"
        return
    st.header("Payment")
    st.write(f"Paying Reservation ID {bill['reservation_id']} — Amount: {bill['amount']:.2f}")
    st.write("**Demo only**: This simulates payment. Do NOT enter real card details here in production.")
    with st.form("payment_form"):
        cc = st.text_input("Card Number (16 digits)", max_chars=16)
        exp = st.text_input("Expiry MM/YY", max_chars=5)
        cvv = st.text_input("CVV (3 digits)", max_chars=3, type="password")
        submitted = st.form_submit_button("Pay Now")
        if submitted:
            if len(cc) == 16 and len(cvv) in (3,4):
                # simulated success
                st.success("Payment successful (simulated).")
                mark_reservation_paid(bill["reservation_id"], bill["amount"])
                # prepare receipt info
                receipt = {
                    "Reservation ID": bill["reservation_id"],
                    "User": st.session_state.user_id,
                    "Amount Paid": f"{bill['amount']:.2f}",
                    "Paid At": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                }
                st.session_state.receipt = receipt
                st.session_state.page = "receipt"
            else:
                st.error("Invalid simulated card details.")
    if st.button("Back"):
        st.session_state.page = "bill"

# ---------- Receipt ----------
def receipt_page():
    show_sidebar_user_info()
    r = st.session_state.get("receipt")
    if not r:
        st.error("No receipt available.")
        st.session_state.page = "welcome"
        return
    st.header("Receipt")
    for k, v in r.items():
        st.write(f"**{k}:** {v}")
    if st.button("Back to Welcome"):
        st.session_state.page = "welcome"

# ---------- Account history ----------
def history_page():
    show_sidebar_user_info()
    user_id = st.session_state.get("user_id")
    df = reservations_for_user(user_id)
    st.header("Account History")

    if df.empty:
        st.info("No reservations yet.")
```

```
    else:
        st.dataframe(df)
        st.write("To cancel a future reservation, enter its Reservation ID below (0 = none).")

        # Allow 0 as default to avoid StreamlitValueBelowMinError
        rid = st.number_input("Reservation ID to cancel", min_value=0, value=0)

        if rid != 0 and st.button("Cancel Reservation"):
            ok, err = cancel_reservation_db(rid, user_id)
            if ok:
                st.success("Cancelled successfully.")
            else:
                st.error(f"Could not cancel: {err}")

    st.markdown("---")  # separator
    if st.button("Back to Welcome"):
        st.session_state.page = "welcome"


```

```

# ================== ROUTER / MAIN ==================
def main():
    st.title("")

    # initialize DB once
    init_db()

```

```

    # session defaults
    if "page" not in st.session_state:
        st.session_state.page = "home"

    page = st.session_state.page

    if page == "home":
        home_page()
    elif page == "about":
        about_page()
    elif page == "signup":
        signup_page()
    elif page == "login":
        login_page()
    elif page == "welcome":
        welcome_page()
    elif page == "reserve_time":
        reserve_time_page()
    elif page == "choose_level":
        choose_level_page()
    elif page == "choose_slot":
        choose_slot_page()
    elif page == "confirm_reservation":
        confirm_reservation_page()
    elif page == "bill":
        bill_page()
    elif page == "payment":
        payment_page()
    elif page == "receipt":
        receipt_page()
    elif page == "history":
        history_page()
    else:
        st.session_state.page = "home"
        home_page()

```

```

if __name__ == "__main__":
    main()
#End of code
```

SQL CODE

```
mysql> create database parking;
Query OK, 1 row affected (0.01 sec)

mysql> use parking;
Database changed
mysql> create table user_details(
    -> user_id varchar(20) primary key,
    -> user_name varchar(50),
    -> user_password varchar(20),
    -> user_addr varchar(100),
    -> vehicle_no varchar(14),
    -> vehicle_type integer,
    -> user_mobile_no bigint);
Query OK, 0 rows affected (0.09 sec)
```

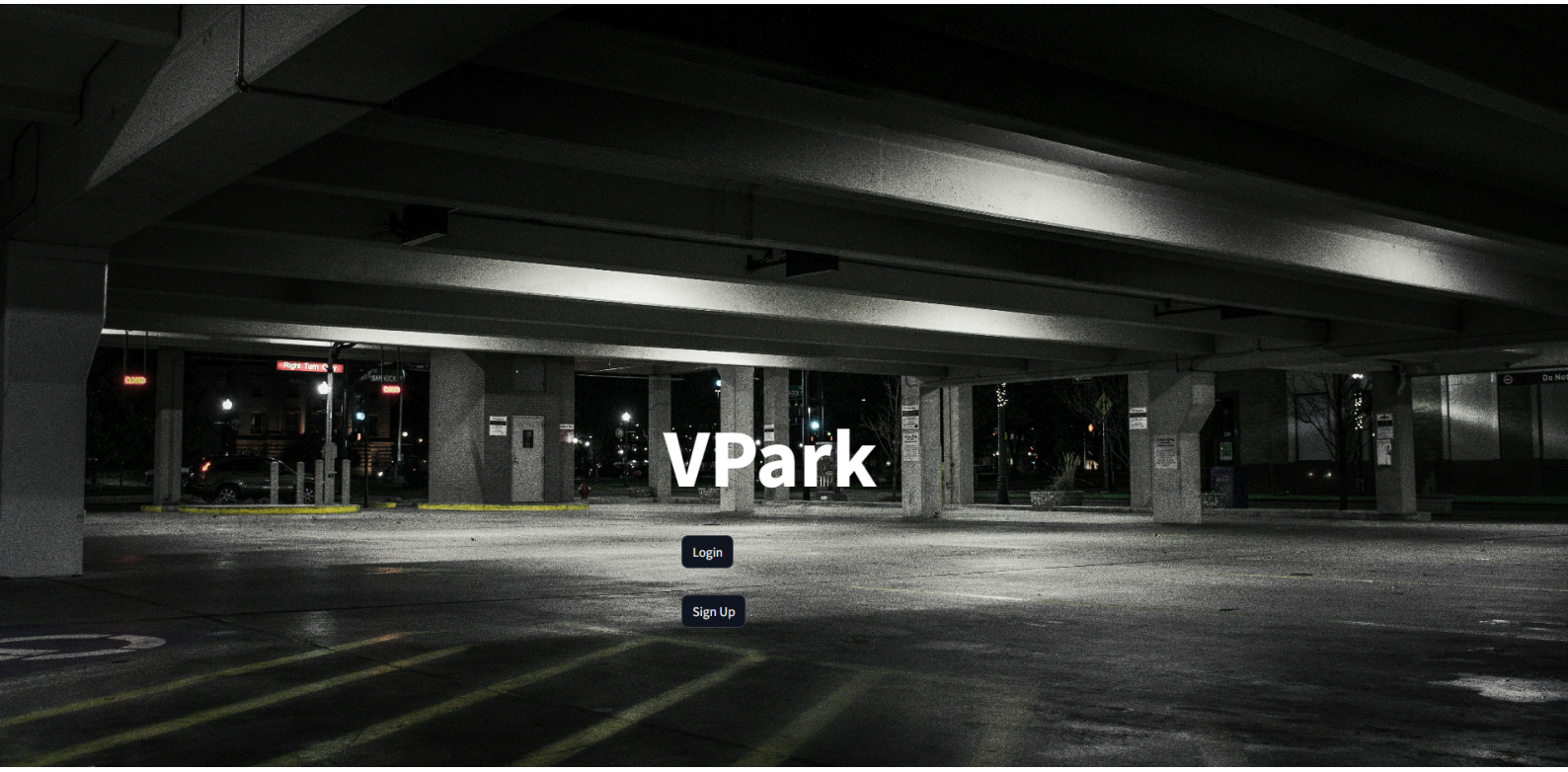
```
mysql> create table spot_check(
    -> user_id varchar(20) primary key,
    -> spot_no integer,
    -> entry_time time,
    -> exit_time time,
    -> level_no integer,
    -> date_of_parking date);
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> desc user_details;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| user_id        | varchar(20)  | NO   | PRI | NULL    |       |
| user_name      | varchar(50)  | YES  |     | NULL    |       |
| user_password  | varchar(20)  | YES  |     | NULL    |       |
| user_addr      | varchar(100) | YES  |     | NULL    |       |
| vehicle_no     | varchar(14)  | YES  |     | NULL    |       |
| vehicle_type   | int          | YES  |     | NULL    |       |
| user_mobile_no | bigint       | YES  |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
7 rows in set (0.05 sec)
```

```
mysql> desc spot_check;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| user_id        | varchar(20) | NO   | PRI | NULL    |       |
| spot_no        | int         | YES  |     | NULL    |       |
| entry_time     | time        | YES  |     | NULL    |       |
| exit_time      | time        | YES  |     | NULL    |       |
| level_no       | int         | YES  |     | NULL    |       |
| date_of_parking | date       | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Output Window:

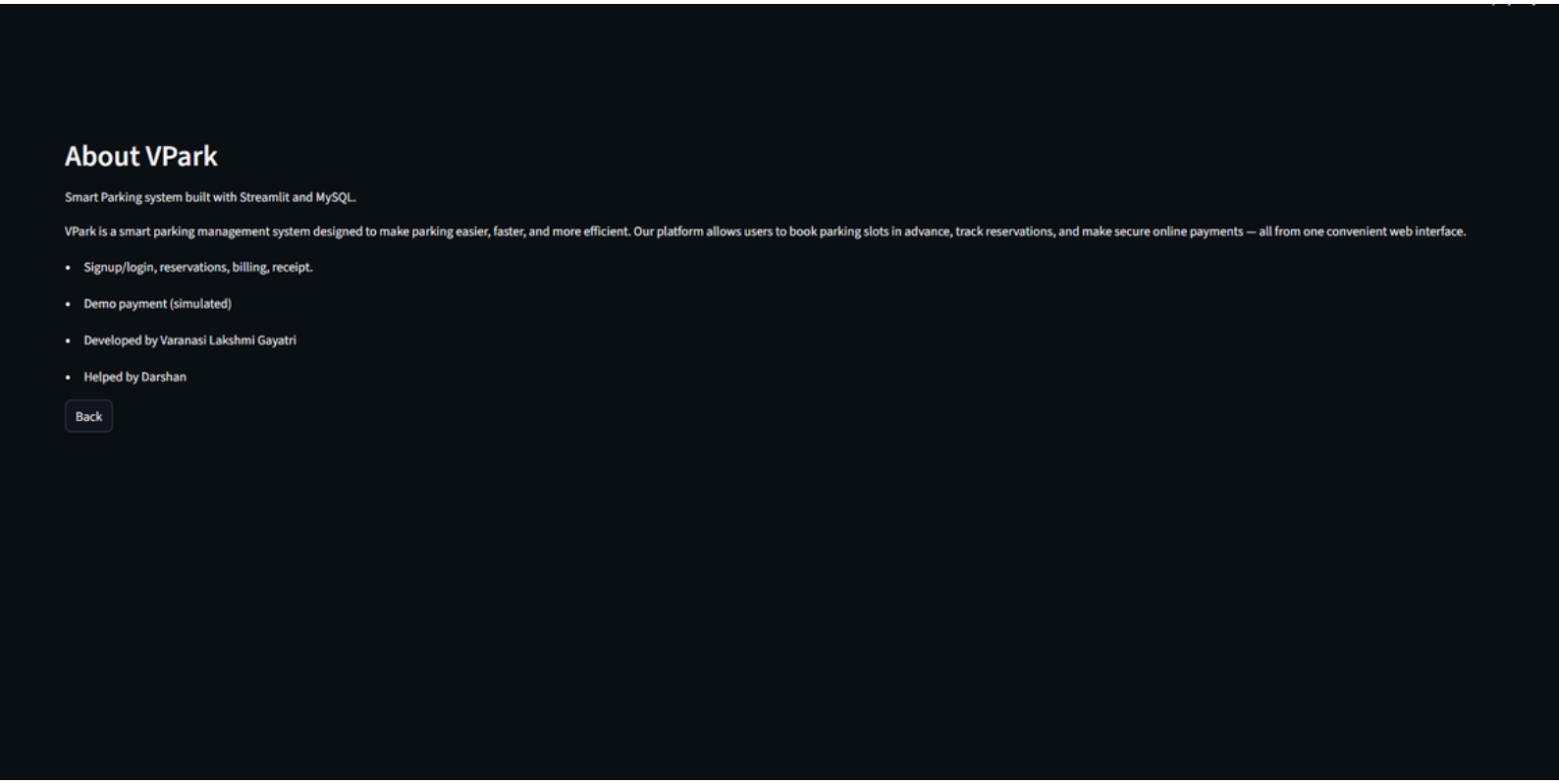
Home Page:



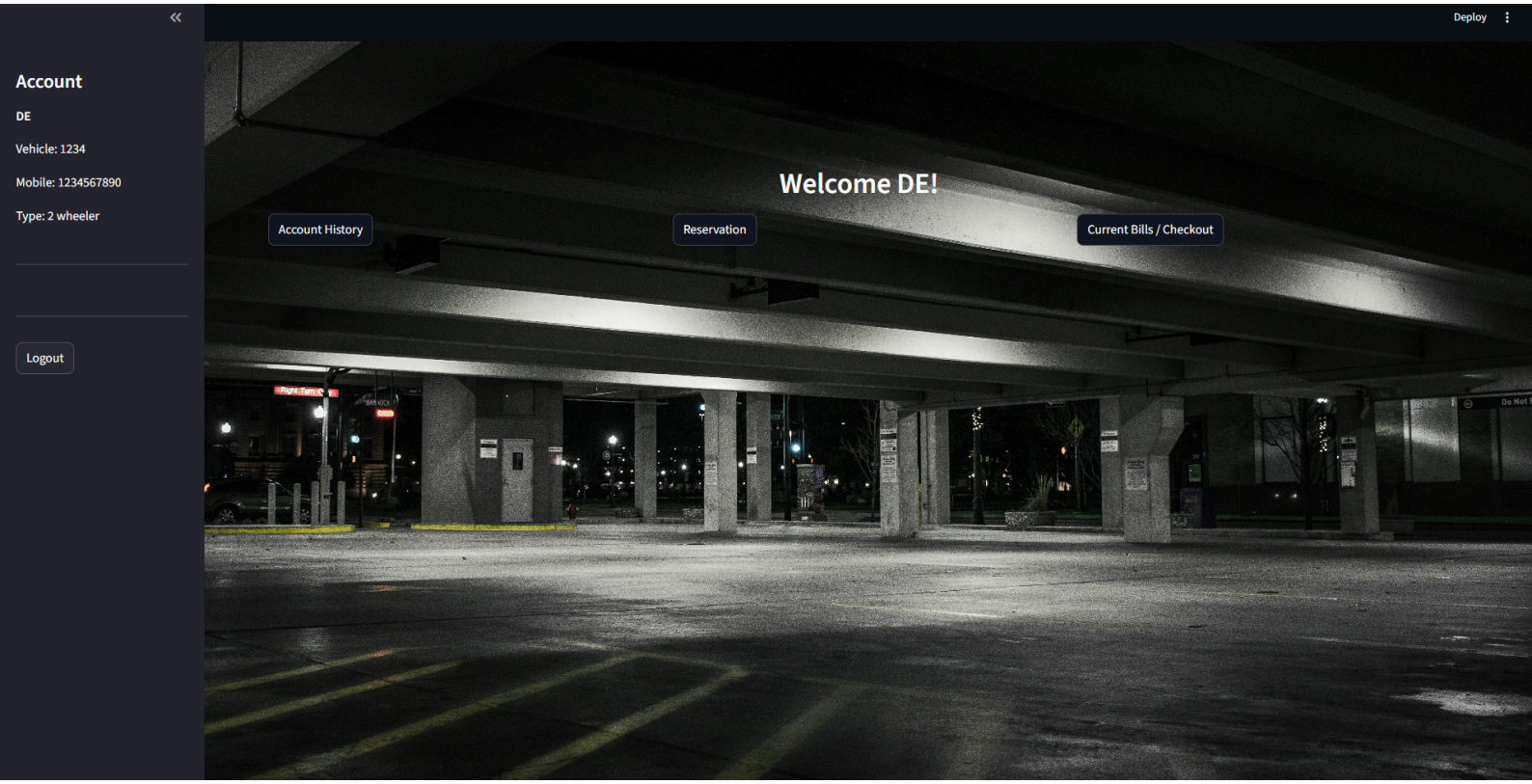
Sign-Up Page:

Log-In Page:

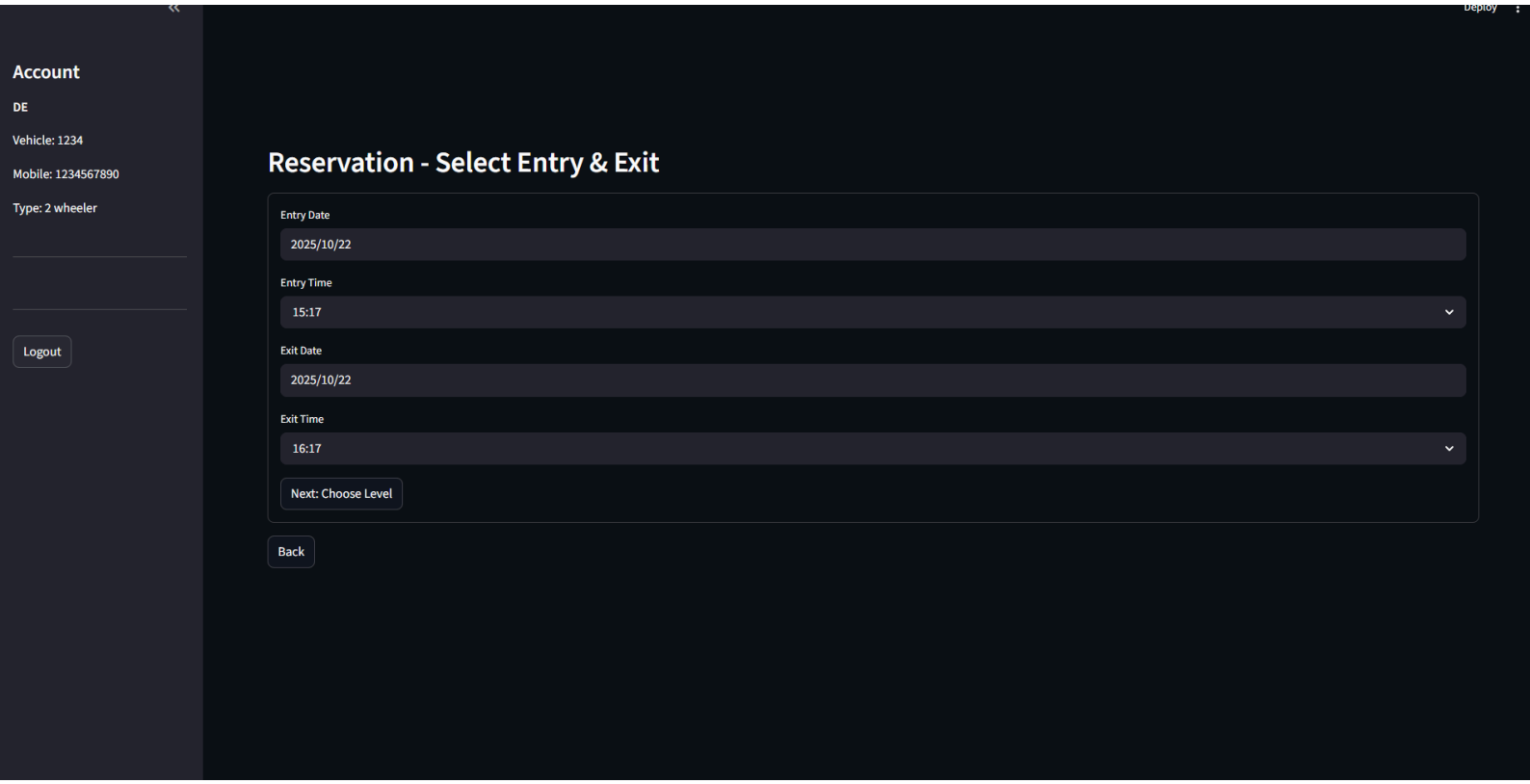
About us Page:



Welcome Page:



Reservation Page:



Account

DE

Vehicle: 1234

Mobile: 1234567890

Type: 2 wheeler

Logout

Choose Parking Level

Level 1 | Level 2 | Level 3

Back

Account

DE

Vehicle: 1234

Mobile: 1234567890

Type: 2 wheeler

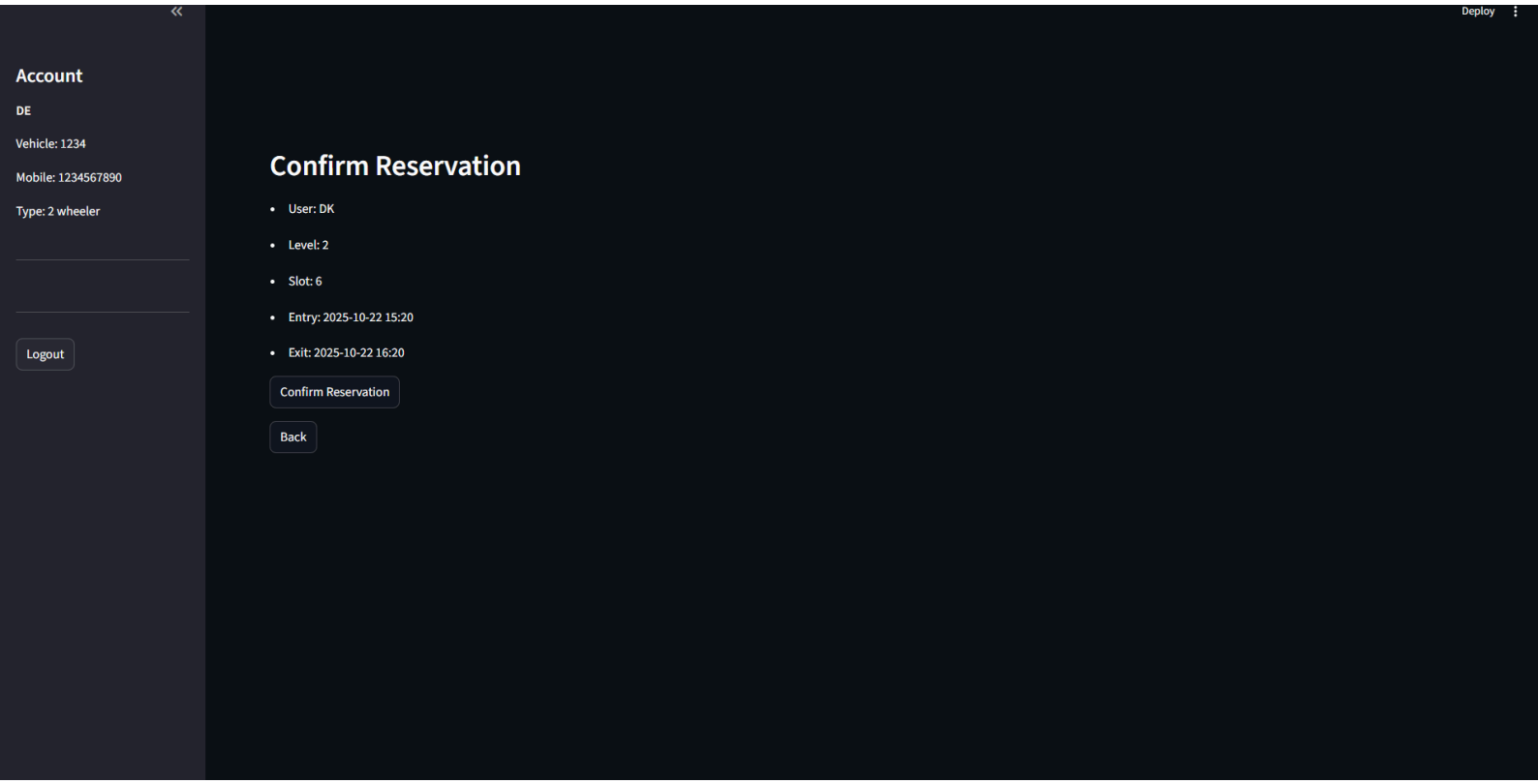
Logout

Choose Slot — Level 2

2025-10-22 15:20 → 2025-10-22 16:20

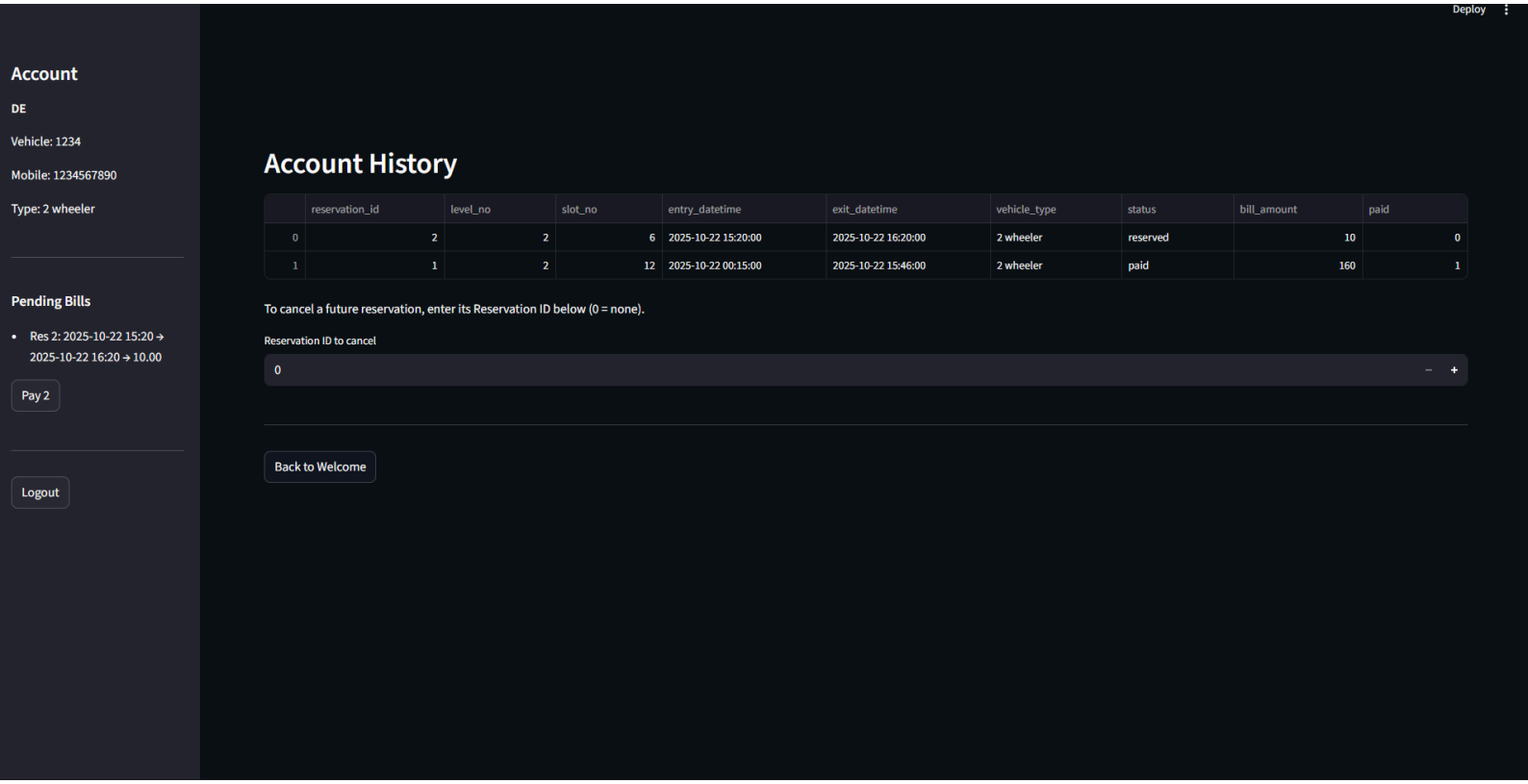
Slot 1	Slot 2	Slot 3	Slot 4	Slot 5
Slot 6	Slot 7	Slot 8	Slot 9	Slot 10
Slot 11	Slot 12 (Used)	Slot 13	Slot 14	Slot 15
Slot 16	Slot 17	Slot 18	Slot 19	Slot 20

Back

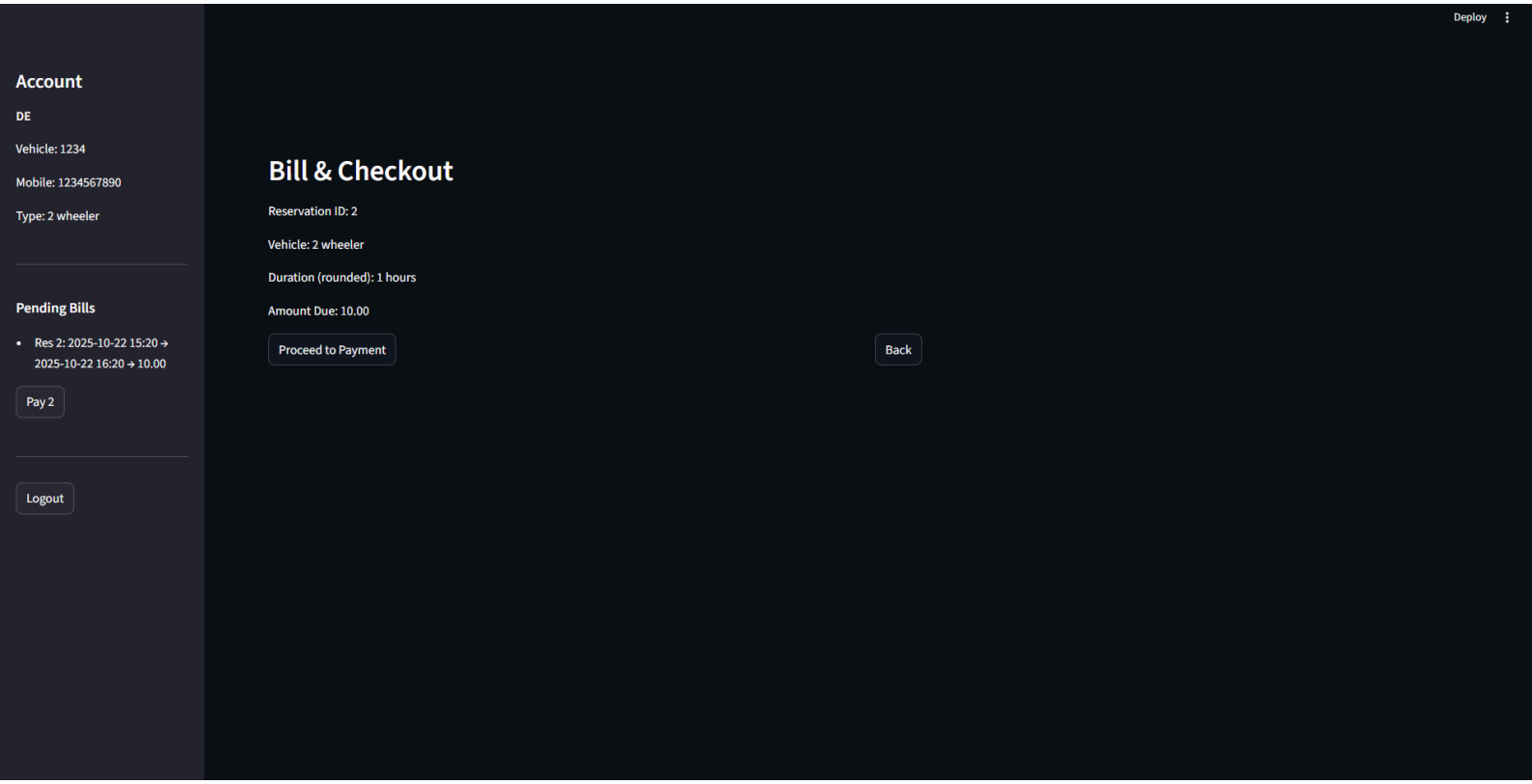

Deploy
Account
DE
Vehicle: 1234
Mobile: 1234567890
Type: 2 wheeler
Logout
Confirm Reservation
User: DK
Level: 2
Slot: 6
Entry: 2025-10-22 15:20
Exit: 2025-10-22 16:20
Confirm Reservation
Back


Account
DE
Vehicle: 1234
Mobile: 1234567890
Type: 2 wheeler
Logout
Confirm Reservation
User: DK
Level: 2
Slot: 6
Entry: 2025-10-22 15:20
Exit: 2025-10-22 16:20
Confirm Reservation
Reservation created.
Back

Account History:



Bill/Checkout Page:



Deploy

Account

DE

Vehicle: 1234

Mobile: 1234567890

Type: 2 wheeler

Pending Bills

- Res 2: 2025-10-22 15:20 → 2025-10-22 16:20 → 10.00

Pay 2

Logout

Payment

Paying Reservation ID 2 — Amount: 10.00

Demo only: This simulates payment. Do NOT enter real card details here in production.

Card Number (16 digits)

Expiry MM/YY

CVV (3 digits)

Pay Now

Back

Account

DE

Vehicle: 1234

Mobile: 1234567890

Type: 2 wheeler

Logout

Receipt

Reservation ID: 3

User: DK

Amount Paid: 10.00

Paid At: 2025-10-22 15:25:29

Back to Welcome

Bibliography

- **Streamlit Documentation**
  Streamlit Inc. (2025). *Streamlit: The fastest way to build and share data apps.*
  Retrieved from https://docs.streamlit.io
- **MySQL Reference Manual**
  Oracle Corporation. (2025). *MySQL 8.0 Reference Manual.*
  Retrieved from https://dev.mysql.com/doc/
- **bcrypt Library**
  PyPI – *bcrypt 4.0.1: Modern password hashing for your software and servers.*
  Retrieved from https://pypi.org/project/bcrypt/
- **Pillow (PIL) Documentation**
  The Python Software Foundation. (2025). *Pillow: Image processing capabilities for Python.*
  Retrieved from https://pillow.readthedocs.io
- **Pandas Documentation**
  McKinney, W. et al. (2025). *Pandas: Data analysis and manipulation library.*
  Retrieved from https://pandas.pydata.org/
- **Python Official Documentation**
  Python Software Foundation. (2025). *Python 3.12 Documentation.*
  Retrieved from https://docs.python.org/3/
- **Image Resources**
  Background and car images sourced from royalty-free image repositories (*Pixabay*, *Pexels*, and *PNGTree*) for demonstration purposes.

Conclusion

In an era of raging parking issues, optimal usage of parking space and the abruptness of parking vehicles are critical factors. Lack of effective parking management creates problems for everyone. Parking systems routinely experience parking-related challenges, especially in urban and metropolitan areas. The most significant of them is the availability of space. Either the operators need help finding parking slots or the capacity needs to be more utilized. Technology-based Parking Management System is an automated solution that provides an advanced and rapid solution, right from an entry in the parking area to the exit. This project does this by providing means to reserve parking spaces and showing all the unoccupied slots to maximize the utility of all the parking spaces.