CECS 490B – Computer Engineering Senior Project II – Team Final Report

Rodijez Agatep - Sanjana Das - Maryith Garzon - Edward Ramos

Team Name: Beach Park

https://beach-park.github.io/Beach-Park/

Note: User accounts and reservations are offline since host must be connected to same internet as user device for proper function. Rest of web-app functions properly

Video Demonstration: https://www.youtube.com/watch?v=Iovayu3T4Dw

Team Members and Biography

Rodijez Agatep is a 22-year-old Computer Engineering Major at California State University Long Beach. He has been a student at CSULB since Fall 2020 and has an expected graduation of Spring 2025. He has been wanting to pursue a career in engineering since 6th grade. Rodijez has been fascinated with technology ever since he was a child. His hobbies include video games and reading.

Sanjana Das is a 21-year-old Computer Engineering Major and minor in Applied Mathematics student at California State University Long Beach. She has been a student since Fall 2021 with an expected graduation of Fall 2024. She wishes to pursue a master's in biomedical engineering as she has always wanted to get in the medical field and work with medical devices. Her hobbies include playing chess and travelling.

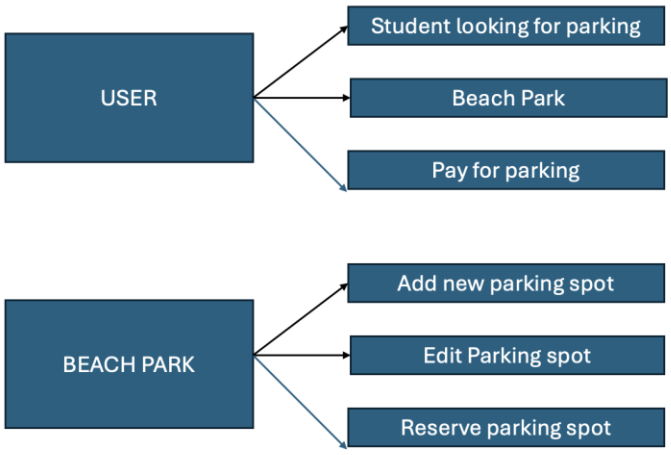
Viviana Garzon is currently a senior student at California State University, Long Beach, pursuing a degree in computer engineering. Throughout her academic journey, she has excelled in coursework covering areas such as software development, hardware design, and Embedded systems. Outside the classroom, Viviana has actively participated in various extracurricular activities related to computer engineering, including hackathons, BootCamps and clubs. Her determination to keep learning drives her ambition to make significant contributions to computer engineering.

Edward Ramos is a Computer Engineering Major at California State University, Long Beach. He has been a student since Spring 2022 with an expected graduation in Fall 2024. He has been tech savvy since he was around 8 years old. He was also curious about how computers worked. In his free time, he loves to play video games and spend time outside.

Project Overview

We students know how hard it is to find parking on campus on time and get to class. Most students end up missing classes or have to park very far that exhaust them by the end of the day. It is important to consider time, health, and management at the same time as doing classes. Therefore, the Beach Park web-app that will help resolve these issues.

Beach Park is an app that will allow users to reserve their own parking spaces via a webserver and web-app. This is designed for CSULB parking department, which in turn will be used by students and faculty. Our project consists of 3 parts: software (webserver; web-app), hardware (sensors and microcontrollers), and miniature (model). The purpose of this project is to reduce the amount of time needed to find a parking space; an increasing problem that many students have faced during their enrollment here at CSULB.



The diagram above shows the actions that the user and the system [software + hardware] will be taking. The user has the choice of searching, reserving, and confirming a parking spot. The system can add, edit, or delete a parking spot. To use the web-app, the user can login with a username and password. Next, they can select an open parking spot from the user interface. They can select an allotted time for the reservation, after that, they can place their vehicle in front of the sensor on the miniature. The web-app and LEDs will update in real time.

Customer Needs

Minimum Needs	Assumed Needs	Unrecognized Needs
• System operates from a Web-Server • Other devices can connect to web-server • Web-App is easy to read/use; has simple UI • App allows user to login • User can input car info into account • App can search for an available parking spot • User can reserve an available spot • User can reserve spot for specified amount of time • User can delete their reserved spot • LEDs show status of parking spot	• Physical Model to simulate parking spaces	• Model has themed environment

• Model has clearly labeled spots/zones		

Design Specifications

Minimum Needs

- System operates from a Web App
    - User can access app from computer or mobile device
    - System requires internet connection
    - Web app has no bugs and crashes

- Other devices can connect to web-server
    - Web-app can be accessed by another device (that is not the host device)

- App is easy to read/use; has simple UI
    - App has simple layout with clear buttons, correct alignment, labeling
    - App language is in English
    - App has no spelling/grammar errors

- App allows user to login
    - User can create account (username/password and email)
    - System saves user accounts
    - System accepts user after successful login
    - System denies user after unsuccessful login

- User can input car info into account
    - User can input license plate number into account
    - Account saves license plate number info
    - User can input car model into account
    - Account saves car model info
    - User can delete or modify plate and model info

- App can search for an available parking space
    - User can only search for an open spot
    - App will update in real time based on sensors

- User can reserve an open spot
    - User can only reserve an open spot (same spot that they just searched)
    - App updates in real time based on sensors

- Use can reserve spot for certain amount of time
    - User selects the MM/DD/YYYY to reserve
    - User cannot select time that is earlier than current time
    - User select from various preset times (30 min, 2 hr, 4 hr, 12 hr)
    - App and LED update once time has elapsed
    - User receives notification that reserve time is finished

- User can delete their reserved spot
    - User can only delete their own reserved spot
    - App updates in real time based on sensors

- LEDs show status of parking spot
    - Available is shown as green
    - Reserved is shown as blue
    - Unavailable is shown as red

- Model has clearly labeled spots/zones
    - Spots correspond to respective sensor and app UI
    - The 2 zones are physically separated

Assumed Needs

- Physical Model to simulate parking spaces
    - Model is able to visually demonstrate how parking system operates

Unrecognized Needs

- Model has themed environment
    - Model has 3D-printed pyramid centerpiece
    - Model is winter themed
    - Model is roughly scaled to a Hot Wheels car

Constraints

- **Time:** This project will have to be completed within the Spring to Fall 2024 Semesters
- **Cost:** The cost of this project and its related materials will be covered by the CSULB reimbursement program.
- **Scope:** Project consists of 3 main sections: Software, Hardware, and Miniature
- **Quality:** Project must be tested thoroughly to ensure no bugs or glitches in final product. Must meet the customer needs and design specifications
- **Resources:** Project may be limited by our knowledge of our hardware and software.

Project Implementation

Software

The microcontrollers are programmed in C and use ultrasonic sensors to measure distances and manage parking spaces. It works by triggering each ultrasonic sensor to measure the distance to an object, like a car. If the measured distance is less than or equal to 10 cm, it assumes a car is present in the parking spot. The program also controls LEDs to show the parking spot's status using different colors: green for empty, red for occupied, and blue for reserved spots.

The program uses three ultrasonic sensors connected to the microcontroller. Each sensor is triggered one at a time to avoid interference. The system waits for the sensor's echo signal, which represents the time it took for the sound wave to travel to the object and back. Using this time, the program calculates the distance and decides the status of the parking spot.

To improve usability, the system allows users to reserve or unreserve parking spots. This is done through UART communication, where the microcontroller receives commands to mark a parking spot as reserved or available. Reserved spots are indicated with blue LEDs, and the system ensures they are not marked as empty, even if no car is present.

The program also includes interrupt handling. When an ultrasonic sensor detects an echo signal, it triggers an interrupt, and the program calculates the distance. Another interrupt is used for UART communication, where it reads user commands and updates the reservation status of the parking spots.

The parking reservation system also integrates a web server programmed in Node.js to display the parking spot statuses in real-time. The web server runs on a Raspberry Pi, which acts as the master device, while the microcontroller functions as a slave device. The server communicates with the microcontroller over UART to request and receive parking status updates.

The web server is designed to handle requests from users and send commands to the microcontroller. For example, if a user wants to check the status of a specific parking spot, the server sends a request to the microcontroller to retrieve the data. The microcontroller responds with the current status of the parking spots, including whether they are empty, occupied, or reserved. This data is then processed by the Node.js server and displayed on a web page, making it accessible to users through any device with internet access.

Node.js is used because it handles asynchronous operations efficiently, which is ideal for real-time systems like this. The server uses WebSocket communication to push updates to the web page whenever new data is available. This makes sure that users can see live changes without needing to refresh the page. For example, if a car enters or leaves a parking spot, or if a spot's reservation status changes, the server updates the web interface automatically.

The system also stores parking data in a Firebase Realtime Database, allowing for centralized management and better scalability. The Raspberry Pi reads data from Firebase, processes it, and sends commands to the microcontroller to update the status of parking spots.

Hardware

For this project, we decided very early in our planning stage that we would utilize HC-SR04 Ultrasonic Sensors and TM4C123GH6PM microcontroller. Two pieces of hardware that we have previously used in our Embedded Systems courses.

The Raspberry Pi was chosen because it is easy to use, powerful, and works well with web servers and databases. It acts as the main device that collects data from the two microcontrollers and shows the parking status in real time. With the Raspberry Pi, users can check parking information easily through a website either from their computers or mobile phones.

Miniature

The model is roughly scaled to a Hot Wheels car; it features 2 parking zones with 3 spots each. A 3D-printed pyramid serves as the centerpiece of this model. There are 3 sensors facing

outwards on the left and right side of the pyramid. Each set of 3 parking spaces is its own zone. They are physically separated by the pyramid. The environment is Winter themed.

Initially, we were going to make a model of the entire lower part of campus (Walter Pyramid and soccer field to the engineering buildings and to the Palo Verde Parking Structures. We then decided to focus more on the pyramid and have the parking surrounding it. We were inspired by the Walter Pyramid on the CSULB campus and Luxor Las Vegas. The theme of the model switched from Fall to Winter to better correspond to the time of our project demonstration. More information about the model (dimensions and layout) are in the *Miniature Model* portion of this report.

Subcomponent Descriptions

Two TM4C123GH6PM Launchpad Development Boards by Texas Instruments.

The TM4C123GH6PM Launchpad contains the ARM Cortex-M4F microcontroller. In our project, the peripherals used are GPIO (General Purpose Input-Output), Timers, UART (Universal Asynchronous Receiver-Transmitter) Serial Communication, and NVIC (Nested Vectored Interrupt Controller). Both microcontrollers utilize the same functionality and is operated at the default 16MHz clock:

GPIO

We utilize the GPIO ports A, B, E and D. Ports A5-A7, A2-A4 and E0-E2'S functionality are to send high level values to the RGB Leds to light up the corresponding color, depending on the state the microcontroller receives information from the HC-SR04 Ultrasonic Sensor.

- Green LED: No car present. This specific parking Spot is available.
- Red Led: Car is present. This specific parking spot is not available.
- Blue Led: This specific parking spot has been reserved.

Ports A5-A7, A2-A4 and E0-E2 correspond sensors, 1, 2 and 3 for the first microcontroller and for the second, sensor 4, 5 and 6 which are parking spots 1, 2 3, 4, 5 and 6 respectfully.

Timers

We use Timer0A, Timer1A and Timer 2A to calculate distance if an object is placed in each of the HC-SRO4 sensors. Each of those timers correspond to sensors 1, 2 and 3 for the first microcontroller and the second, 4, 5 and 6. When the microcontroller detects the rising edge coming from the echo pin signal of the HC-SR04 sensor, then the timer corresponding to the sensor will start and stop counting when the falling edge is detected. Then we take the timer lapse value and use it to calculate the distance using this physics equation:

Distance = Time * Speed

Distance = (Timer Lapse* Machine Cycle Length for a 16MHz clock * Speed of Sound)/2

The machine cycle length in our case is T = 1/16MHz = 0.0625 microseconds and we included it because we consider the clock cycle the two microcontrollers are operating at.

The speed of sound traveling through air at 68 Faren height is 0.0343 c/s.

We divide it in 2 because the measured time accounts for the sound wave traveling to the object and back.

Nested Vectored Interrupt Controller (NVIC)

We use this system-level peripheral to detect rising edges coming from the echo pin of the HC-SR04 sensors. We activate the interrupt for Ports B6, B7 and B0 with priority 3 and to detect both edges. Each of those ports correspond to sensors 1, 2 and 3 for the first microcontroller and sensors 4, 5 and 6 for the second microcontroller.

In the Port B handler, if it were to detect rising edges coming from the echo pins corresponding to ports B6, B7 and B0, it would activate the timer to count the rising edge and stop counting if it detects the falling edge. This is how we are able to get the timer lapse results for both edges and is crucial for calculating the distance.

UART (Universal Asynchronous Receiver-Transmitter) Serial Communication

The two launchpad's microcontrollers primarily read information sent by the ultrasonic sensor via the echo pin and processes the car's distance. Additionally, the microcontroller is the slave and sends out distance information in front of a particular sensor when requested. We use UART2 where ports D6 is the receiver (RX) and D7 is the transmitter (TX). When the master (Raspberry Pi) requests specific available parking information, the launchpad's microcontroller sends the information is has regarding the status of the parking spot. In addition, if a

reservation has been made at a specific parking spot, the slave sends that information, and the microcontroller sets that reservation status for the parking spot. Both microcontrollers use the same UART2 and operate at baud rate of 115,200 for fast data communication.

HC-SR04 Ultrasonic Sensor

- Operates on a 5V power supply
- Reads trigger pulses sent out by the microcontroller and sends an echo signal.
- Echo and trigger signals are connected to the tm4c123's GPIO pins.

The functionality of the ultrasonic sensor is to detect an object, in this case the car, in front of the sensor. When the microcontroller sends a 10us high signal to the sensor, it sends out the signal and captures the echo reflected from the object. The echo signal is sent to the microcontroller to calculate the distance

UART-to-Ethernet Converter

- Converts UART to ethernet signal.
- Used for long distance network communication
- Powered by 3.3V supplied from one of the microcontrollers.
- UART'S Tx and Rx GPIO pins from the microcontrollers are connected to the converter

This converter is interfaced with the two microcontrollers and sends processed data to the Raspberry Pi

- Hosts the web server to display parking spots availability to users.
- Reads signals from the UART-to-Ethernet converter.
- It is the master and requests parking spots availability per each parking spot.

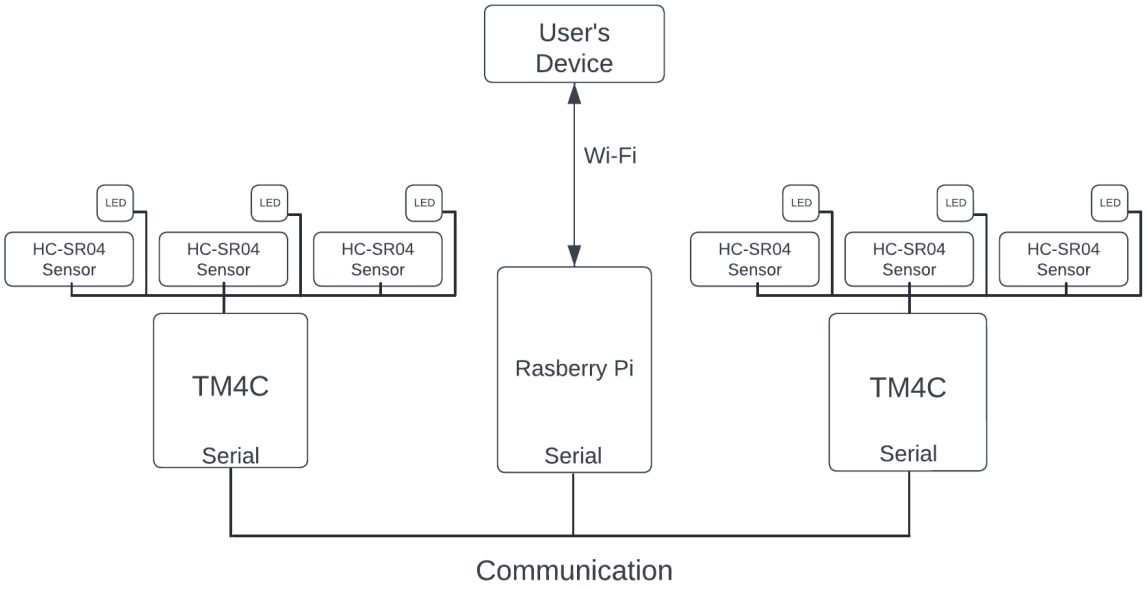
Raspberry Pi will request parking spot availability per each parking spot and uploads the data to the web server for users to see.

Web Application

- Has GUI
- Users can create accounts
- Users can reserve parking spots.
- Reads information sent by the raspberry pi

The web application has a GUI (Generic User Interface) for easy navigation and displays parking spots availability for users to see. If there is an available spot, the registered user can reserve that spot for the amount of time.

Block Diagram



The above block diagram shows a general overview of the structure of our hardware. The user's device will connect with our web-server via Wi-Fi connection. The Raspberry Pi is connected to the 2 TM4Cs; each TM4C is connected to 3 HC-SR04 sensors and 3 LEDs

Overall System Functional Block Diagram(s)

HC-SR04 Ultrasonic Sensor

Signal Name	Signal description	Signal type	Voltage Level	Current Requirements	Operation
Trigger Pin	Input to send ultrasonic signals	Digital	5V	<15mA	Generates ultrasonic pulse
Echo Pin	Output to receive	Digital	5V	<15mA	Output signal

	signal duration				
VCC	Power supply	Power	5V	15mA	Provides power to the sensor
GND	Ground	Power	0V	-	Common ground with microcontroller.

TM4C Microcontroller

Signal Name	Signal description	Signal type	Voltage level	Current Requirements	Operation
Input Pins	Receives echo signals	Digital	5v	<1mA	Captures timing data
Output Pins	Sends trigger signals	Digital	5v	<1mA	Sends signals to sensors
Serial TX	Sends data to raspberry pi	Digital	3.3V	<1mA	Sends data to raspberry pi
Serial RX	Receives data from raspberry pi	Digital	3.3V	<1mA	Receive commands
VCC	Power supply	Power	3.3V	Based on usage	Power the microcontrollers
GND	ground	Power	0V	-	Common ground with sensors and pi

TM4C Microcontroller

Signal Name	Signal Description	Signal Type	Voltage	Current Requirement	Operation
Input Pins	Receives Echo signals from sensors	Digital	5V	< 1mA	Capture timing
Output Pins	Send trigger	Digital	5V	< 1mA	Send signals

Serial TX	Transmits data to Raspberry Pi	Digital	5V	< 1mA	Send data
Serial RX	Receives commands from Raspberry Pi	Digital	5V	< 1mA	Receive commands
VCC	Power Supply	Power	5V	Based on usuage	Powers the microcontrollers
GND	Ground	Power	5V	-	Common ground with sesnors

Raspberry Pi

Signal Name	Signal Description	Signal Type	Voltage	Current Requirement	Operation
Serial RX	Receives data from TM4Cs	Digital	3.3V	<1mA	Reads processed sensor data
Serial TX	Send data to TM$Cs	Digital	3.3V	<1mA	Sends configuration commands
Wifi Transmit	Transmit data	Digital	3.3V	Based on wifi usage	Shares data with user interface
Wifi Receive	Receive commands from user device	Digital	3.3V	Based on wifi	Processes user commands
VCC	Power Supply	Power	5V	~2.5A	Powers the Raspberry Pi and peripherals
GND	Ground	Power	0V	-	Common ground with

					all components

User Device

Signal Name	Signal Description	Signal Type	Voltage Level	Current Requirements	Operation
Wi-Fi Data Rx	Receives data from Raspberry Pi	Digital	3.3V	Based on usage	Displays system data
Wi-fi	Sends commands to raspberry pi	Digital	3.3V	Based on usage	Send control commands

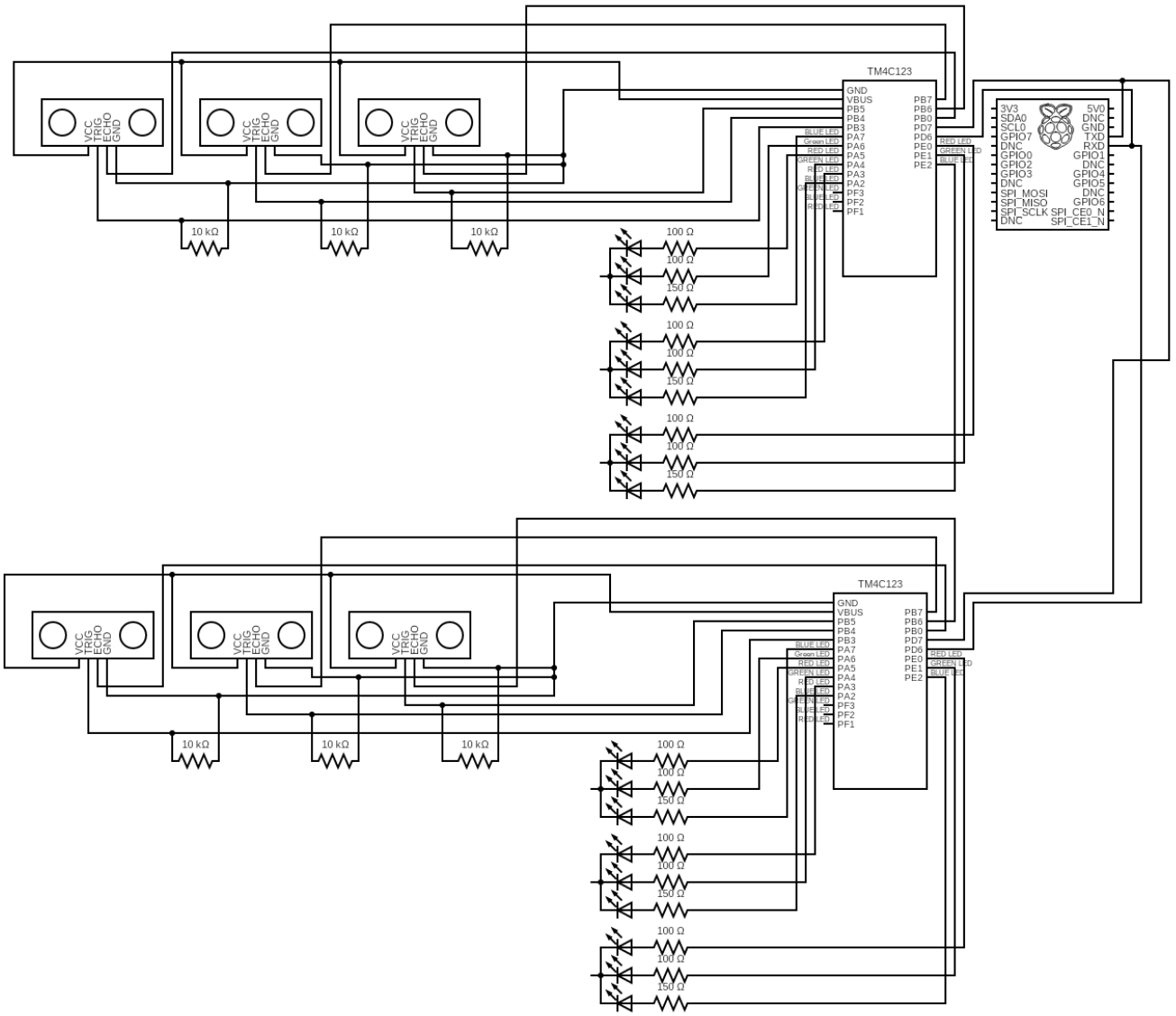
Power Management

The smart car parking reservation system utilizes two microcontrollers and one Raspberry Pi. The TM4C123 microcontrollers, produced by Texas instruments under the launchpad brand, are powered by a 5V supply from the computer's USB port. This power is also distributed to the six ultrasonic sensors connected to the microcontrollers.

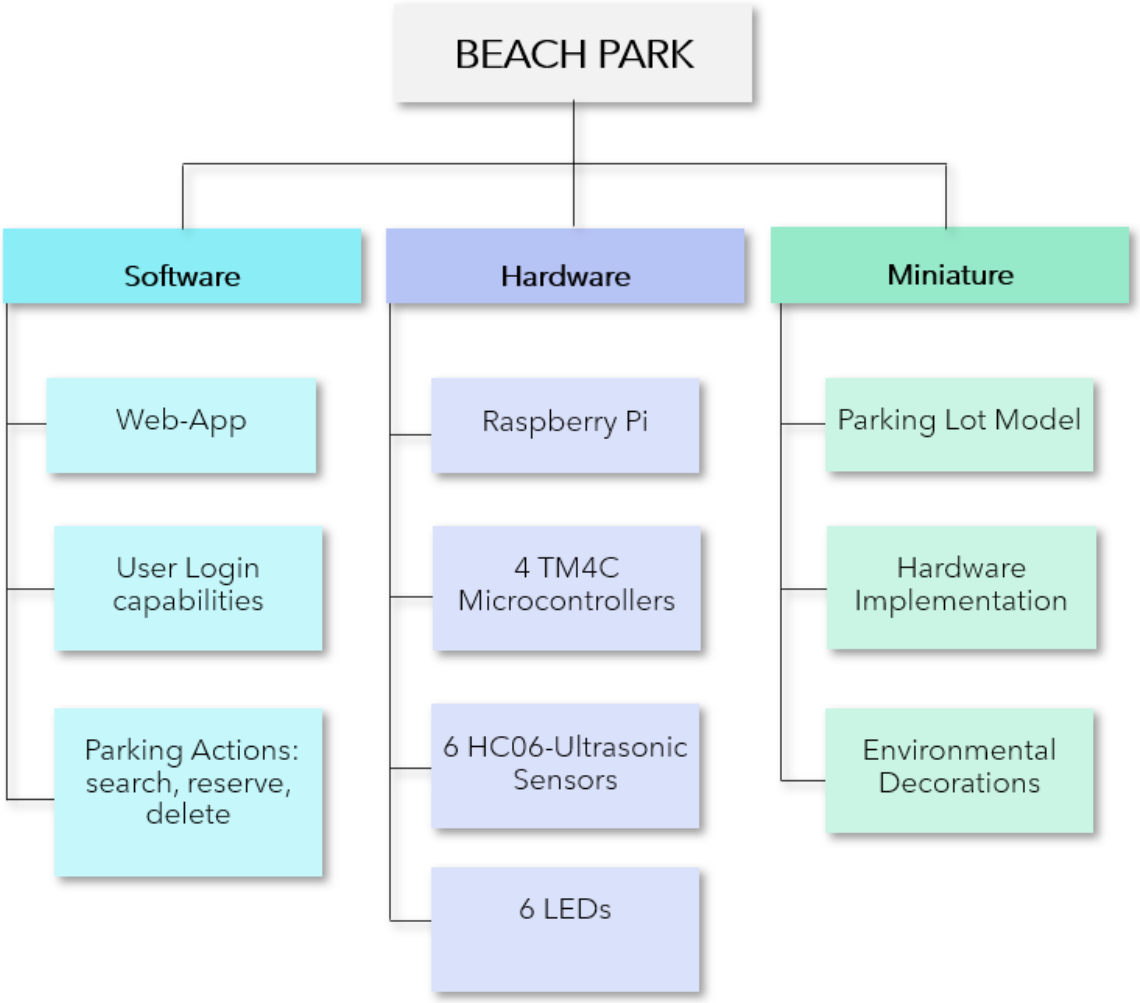
Additionally, the two microcontrollers are connected to the UART-to-Ethernet converter, which receives 3.3V supply provided by the microcontrollers.

The Raspberry Pi is powered independently by a 5v supply produced from its dedicated power adapter.

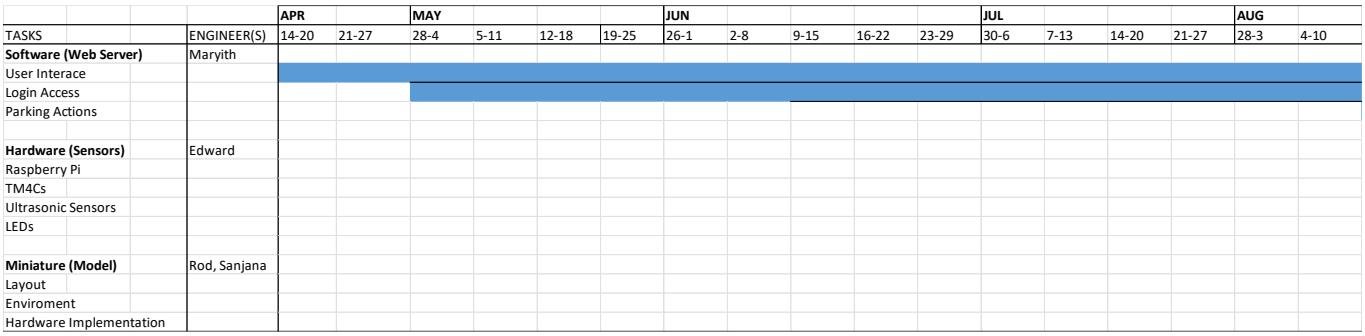
Complete Schematic



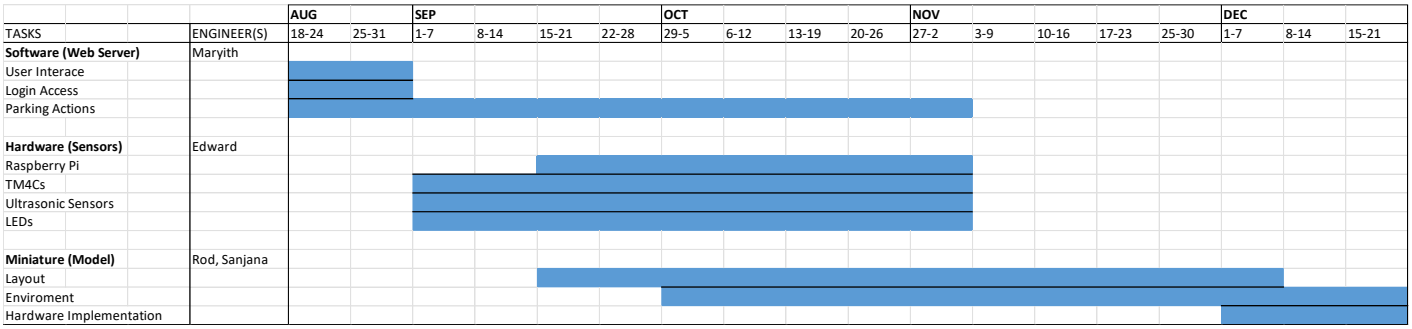
Work Breakdown Structure



Gannt Chart



TASKS	ENGINEER(S)	APR 14-20	APR 21-27	MAY 28-4	MAY 5-11	MAY 12-18	MAY 19-25	JUN 26-1	JUN 2-8	JUN 9-15	JUN 16-22	JUN 23-29	JUL 30-6	JUL 7-13	JUL 14-20	JUL 21-27	AUG 28-3	AUG 4-10
Software (Web Server)	Maryith																	
User Interace																		
Login Access																		
Parking Actions																		
Hardware (Sensors)	Edward																	
Raspberry Pi																		
TM4Cs																		
Ultrasonic Sensors																		
LEDs																		
Miniature (Model)	Rod, Sanjana																	
Layout																		
Enviroment																		
Hardware Implementation																		



TASKS	ENGINEER(S)	AUG 18-24	AUG 25-31	SEP 1-7	SEP 8-14	SEP 15-21	SEP 22-28	OCT 29-5	OCT 6-12	OCT 13-19	OCT 20-26	NOV 27-2	NOV 3-9	NOV 10-16	NOV 17-23	NOV 25-30	DEC 1-7	DEC 8-14	DEC 15-21
Software (Web Server)	Maryith																		
User Interace																			
Login Access																			
Parking Actions																			
Hardware (Sensors)	Edward																		
Raspberry Pi																			
TM4Cs																			
Ultrasonic Sensors																			
LEDs																			
Miniature (Model)	Rod, Sanjana																		
Layout																			
Enviroment																			
Hardware Implementation																			

5 Demo Contract

Demo 1 : “One Parking Sensor”

Objective: We are going to demo one sensor to work as they will be placed in front of each parking lot.

Verification: We will demo one sensor that detects when there is an object present, and an LED will light up.

Measurable Outcome: The value would keep changing based on the placement of the object.

Demo 2: “3 Sensors with Web Server and Miniature Layout”

Objective: We are going to demo 3 of the sensors with the web server. The web server will know if the car is on the spot. We will also begin development of the miniature model that are parking system will be placed in (CSULB campus in an Autumn setting)

Verification: The sensor will communicate with the Raspberry PI to send the data of the available parking spots.

Measurable Outcome: When there is a successful detection of a car that's in a parking space via the sensor, the web app will display accurate real-time display of parking availability with a response of less than 2 seconds. We will also show a general layout of the miniature that we will be building (and the materials that will be used).

Demo 3: "User Interaction on the Web App interface"

Objective: To develop and test the web app interface that will allow users to make reservations and update the parking spot availability.

Verification: This test will allow users to interact with the web app. Users will be able to open the map which shows where the parking lot is located and view the status of the parking spots. The web app will be updated based on the users' interactions.

Measurable Outcome: The demo will show communication with the web application. The updates will be reflected within 2 seconds. The reservation system will be in the state where it will maintain its updates which will prevent double booking of the parking spaces.

Demo 4 : "Parking Lot Model, six sensors and web app"

Objective: We will demonstrate cars parking in real-time. The web app will be shown for parking availability. We will have 60% of the miniature finished.

Verification: There will be a model of a parking lot. They each contain ultrasonic sensors. Each sensor will detect a car that is in the parking spot. Once it detects one, it will communicate with the Raspberry PI to update the parking availability.

Measurable Outcome: There will be accurate detection and real-time statuses of parking availability on the web app for all six sensors in the parking lot model. Each sensor will correctly identify the presence or absence of a car, and the web app will reflect these changes.

Demo 5 : Beach Park

Objective: A majority of our BeachPark system will be completed (software, hardware, and miniature)

Verification: This demo will show all 6 sensors in the miniature, along with the web server.

Measurable Outcome: All 6 sensors will be working properly with the web app; the miniature will be near completion

Similar Products

There are similar projects that will help us understand what we are supposed to do for our project. But most projects available on the internet are based on the use of RFID but our project will be based on using ultrasonic range sensors.

The four similar products that BeachPark is being compared to is: LAX Official Airport Parking Website, JustPark, ParkMobile, and SpotHero. LAX is the dedicated website to reserve parking at the Los Angeles International Airport. The other 3 are third-party websites/apps that are used to reserve parking at select locations. Parking apps utilize systems such as cameras, sensors, and cloud networks to monitor reservations within their lots. This is very similar to BeachPark, which uses sensors and a Web-Server.

Note: Click on each Product Title to get linked to their website

	Product Name			
	(LAX) Official Airport Parking Website	JustPark	ParkMobile	SpotHero
Similarities (all shared compared with BeachPark)	Able to access from online website (downloaded app not required) Able to access both computer and mobile device Ability to sign up to create account Licenses plate registration Parking Actions: search, reserve, cancel, delete spot Reserving spots based on an allotted time Ability to check current status of spots Misc. info (terms of service, about us, contact us)			
Differences (all shared compared	User pays with credit/debit card			

with BeachPark)				
Differences	Website only UI supports multiple languages Can login as a guest Real-time parking map Includes shuttle transportation	Dedicated mobile app Crowdfunded Users can rent out their own driveway	Dedicated mobile app	Dedicated mobile app Can reserve spots monthly

Societal Environmental Impact and Sustainability

There are several ways this system can be used for other purposes:

- **Reduced traffic congestion:** Time that vehicles are spent idling is decreased, which improves driver's experience

- **Decreased Fuel Usage:** Drivers will use less fuel due to the more efficient way of finding parking

- **Energy Efficiency:** Our parking system (if scaled to real life) can utilize solar panels to be more energy efficient.

- **Improved Air Quality:** Vehicles will emit less carbon emission, leading to a healthier environment and less risks for health issues

- **Multi-campus Adoption**: Once this system is implemented at CSULB, the parking management system can be adopted at other universities that face similar parking challenges. This would allow for broader implementation across multiple campuses, contributing to more sustainable solutions on a larger scale.

- **Citywide parking solutions:** The technology and infrastructure developed for the parking management system can be extended to address parking challenges in urban areas beyond university campuses. Cities can adopt and customize the system to manage parking in city centers, residential neighborhoods, and commercial districts, thereby improving traffic flow and reducing congestion citywide.

- **Commercial Parking Facilities:** Private parking operators and commercial establishments can utilize the parking management system to optimize the utilization of their parking facilities. By providing real-time information on available parking spaces and enabling online reservations, businesses can enhance the convenience and accessibility of their parking services for customers.

- **Data Analysis and Insights:** The data collected by the parking management system, such as parking utilization patterns, traffic flow, and user behavior, can be analyzed to derive valuable insights for urban planning, infrastructure development, and transportation policy. These insights can inform decision-making processes aimed at improving overall mobility, sustainability, and livability in urban environments.

Ethical Considerations

- **Sustainability:**
    - **Economic:**
        - Will the implementation of this system lead to an increase in the cost of a parking permit?
        - Web-App must properly function on a wide range of devices

    - **Social**:
        - The most important thing to realize is that some students may not want to use this system. Because of this, using this system will be optional, not mandatory. Each parking zone could designate half of its spots to reserve, and the other half as first come first serve.

    - **Environmental:**
        - System must function under different weather conditions.

- **Privacy:** Web-App must be secure to ensure that personal information is not available to other users.

- **Transparency:** If the system goes offline, a notification should be sent to students.

- **Safety:** Hardware must be located in secure area to reduce chance of injury

Possible Issues/Downsides of This Product

- **Increased Power Usage:** The campus will see an increase in power usage due to the system and the sensors/lights at each parking spot. As mentioned in the ethics section, the campus may increase our permit pass price even higher to compensate for this power usage.

- **Possibility of Software Outages:** The parking system could go offline for an undetermined amount of time. This will cause the app and system to not function; making the web-app unavailable.

- **Possibility of Hardware Damage:** The sensors or wiring may become damaged or defective, leading to errors in the parking system. Extended periods of time and severe weather conditions may contribute to this problem.

- **Increased Permit Prices:** The cost of parking permits may increase due to the cost of running the BeachPark system.

Circuit Simulations, Verifications and Modeling

Circuit Simulations

- Sensor – The ultrasonic sensor sends signals
- Microcontroller simulations – The TM4C microcontroller through the Keil app.
- Communication Testing – Signals between the TM4C controller and raspberry pi
- Power Supply – Each module can be analyzed by the right amount.

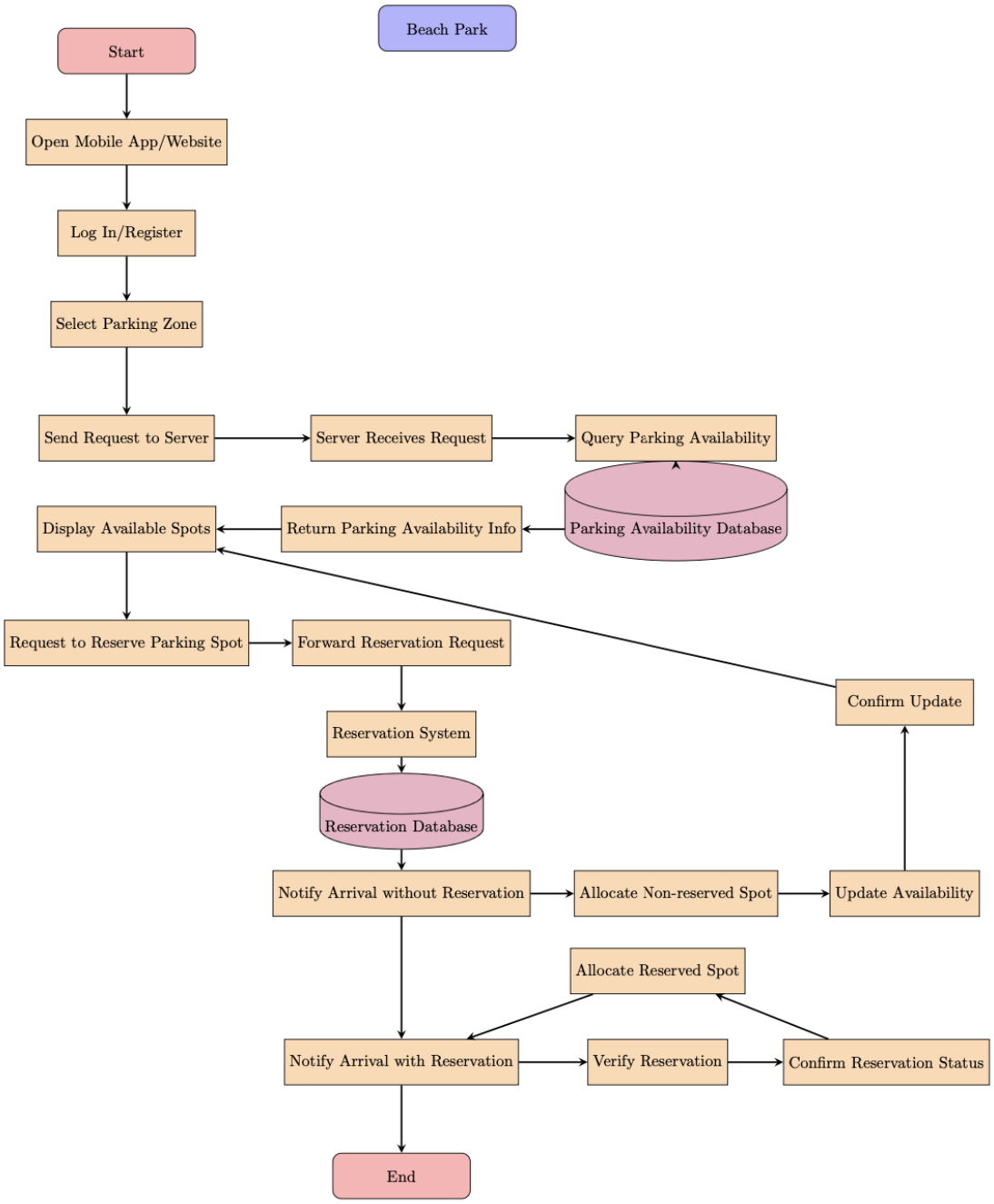
Verification

- The verification was done using oscilloscope. (below image is the UART signal being measured)

Modeling

- The block diagram represents the system into functional modules, showing connections and data flow.

Software Flowchart



Software Flowchart Explanation:

1. Start → User enters credentials.

- **Start**: The flow begins when a user accesses the application through the login or sign-up page.
- **Authentication**:

-The user provides their credentials on the login page.
-Credentials are validated via the /login API endpoint.
-If successful, the system grants access to the dashboard; otherwise, an error message is displayed.

2. Call / login API.

- **Welcome Screen**:
  -Displays a personalized welcome message using the stored username.

- **Parking Spot Reservation**:
  -The user inputs details such as car model, license plate, reservation date, and duration.
  -The system checks the availability of parking spots using Firebase.
  -Upon successful reservation, the system updates the backend database and Firebase.

3. Reservation Management

- If credentials are valid → Store userId and userName locally → Redirect to the dashboard.
- **View Reservations**:
  -The dashboard fetches and displays all reservations associated with the logged-in user via the /reservations/:userId API.

- **Cancel Reservation**:
  -The user selects a reservation to cancel.
  -The system sends a cancellation request to the /cancel API.
  -Upon confirmation, the system updates the backend database and marks the parking spot as available in Firebase.

4. Integration of Google maps

- The Google Maps container in the application displays predefined parking locations around the campus.

5. Error Handling and Feedback

- Handles login errors → invalid credentials

- Displays user-friendly error messages for reservation failures or backend connectivity issues.

Software Specification

- **Frontend**:
  HTML, CSS (Bootstrap), and JavaScript for dynamic and responsive UI.

- **Backend**:
  Node.js server with RESTful API endpoints.
  MongoDB for user and reservation data management.

- **Database**:
  Firebase Realtime Database for parking spot availability.

Framework and Libraries used

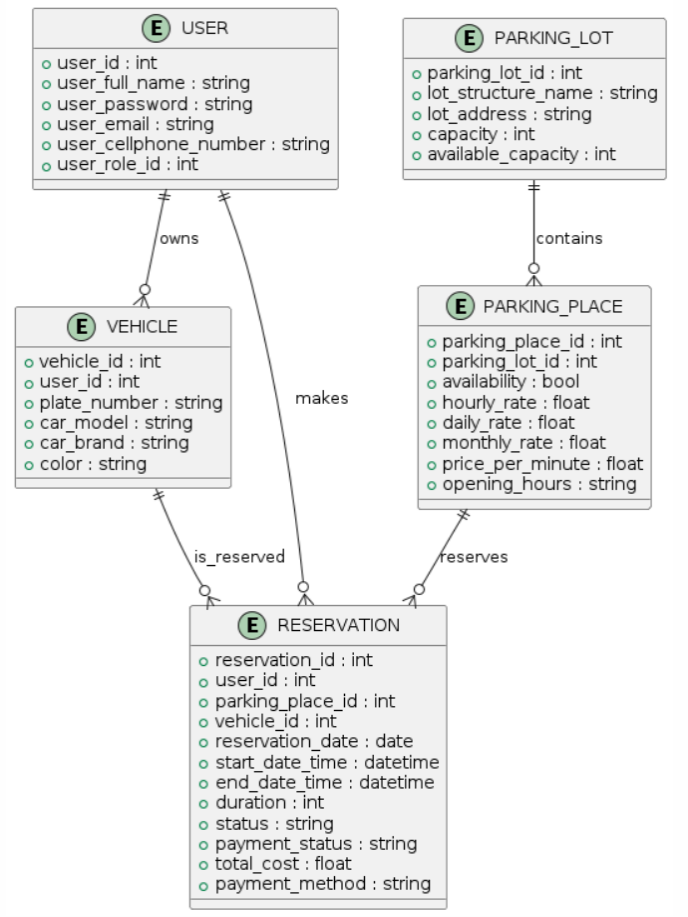
- **Bootstrap**:
  - A front-end framework for building responsive and mobile-first web designs.
  - Used for styling, layout, and ensuring responsiveness.

- **Firebase**:
  - A backend-as-a-service (BaaS) platform.
  - Provides a Realtime Database for dynamic updates of parking spot availability.

- **Node.js**:
  - A JavaScript runtime for building the backend server.
  - Handles API requests for reservations and cancellations.

- **Express.js** (likely used with Node.js):
  - A minimal and flexible Node.js web application framework.
  - Facilitates API creation and server-side routing (though not explicitly mentioned, it is commonly paired with Node.js).

Entity Relationship Diagram



ERD Explanation:

- **User**:
  → Represents individuals using the system to make reservations.
- → Attributes: vehicle_id, user_Id , plate_number, Car_model, Car_brand

- **Parking Lot**:
  → Represents the physical parking structures around campus.
  → Attributes: parking_lot_id, lot_structure, lot_address, capacity, available_capacity

- **Parking Space**:
  Represents individual parking spots within a lot.
  → Attributes: parking_place_id, parking_lot_id, availability, hourly_rate, daily-rate, monthly_rate, price_per_minute, opening_hours.
- **Vehicle**:
  → Represents a user's registered vehicle used for parking.

→ Attributes: venicle_id, user_id, plate_number, car_model, car_brand, color.

- **Reservation**:
  → Tracks parking reservations made by users, linking them to specific vehicles and spaces.
  → Attributes: reservation_id, user_id, parking_place_id, vehicle_id, reservation_date, start_date_time, end_date_time, duration,status, payment_status, total_cost, payment_method.

Software Verification & Validation

Software Verification

Activities Conducted:

1. Code Reviews:

2. Regular reviews of HTML, CSS, JavaScript, and back-end Node.js code ensured adherence to best practices and identified any logic errors.

   GitHub repositories were used to manage code changes, track revisions, and maintain an organized development history.

3. Pull requests and code reviews on GitHub ensured that every change adhered to coding standards before merging into the main branch.

Software Validation

Activities Conducted:

1. Integration Testing:
   Verified interactions between the front-end React components, back-end Node.js services, and databases (MongoDB and Firebase).

2. System Testing:
   The deployed application was tested using the GitHub-hosted version to simulate real-world scenarios.

Cross-browser and device testing ensured responsiveness and usability across different environments.

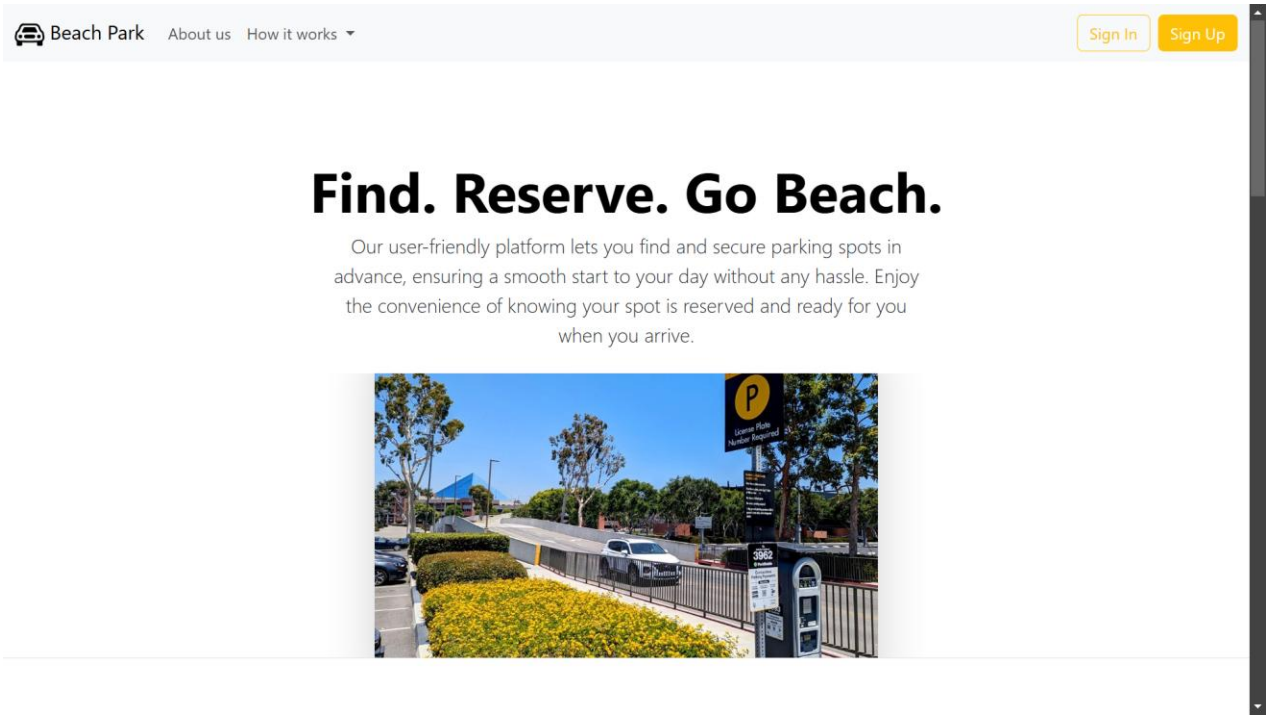
3. Continuous Deployment :
   Using Kubernetes, updates to the application were automatically deployed to the cluster with zero downtime.

Tools Used:

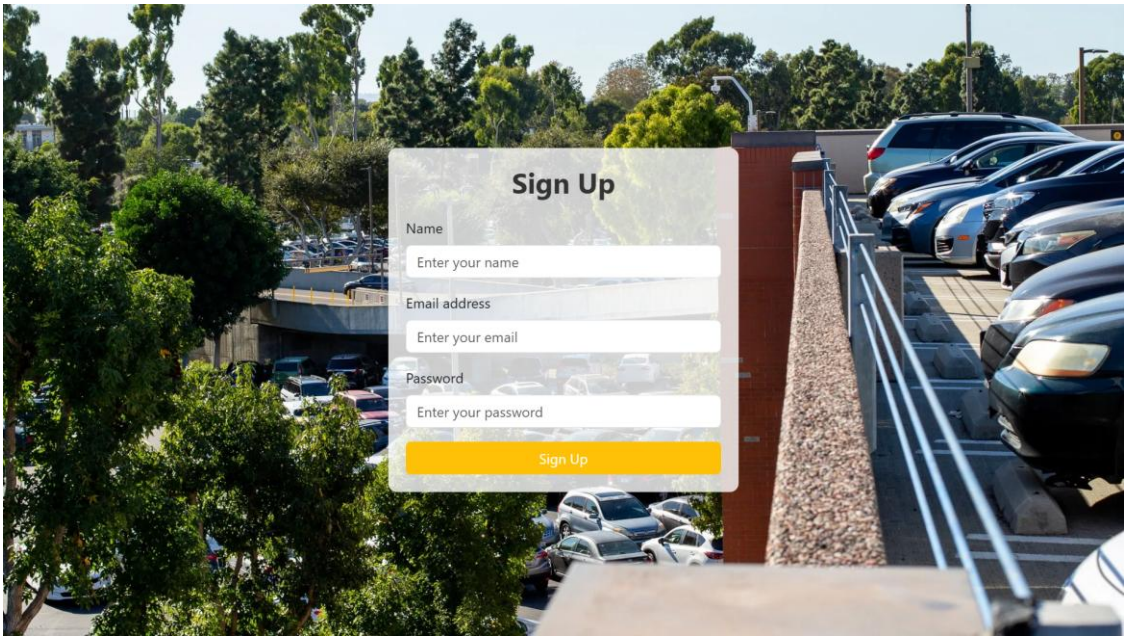
- GitHub: Version control, peer reviews, CD for automated testing.
- Kubernetes: Real-world deployment, scalability testing, and fault-tolerance validation.

Beach Park Web-App Screenshots

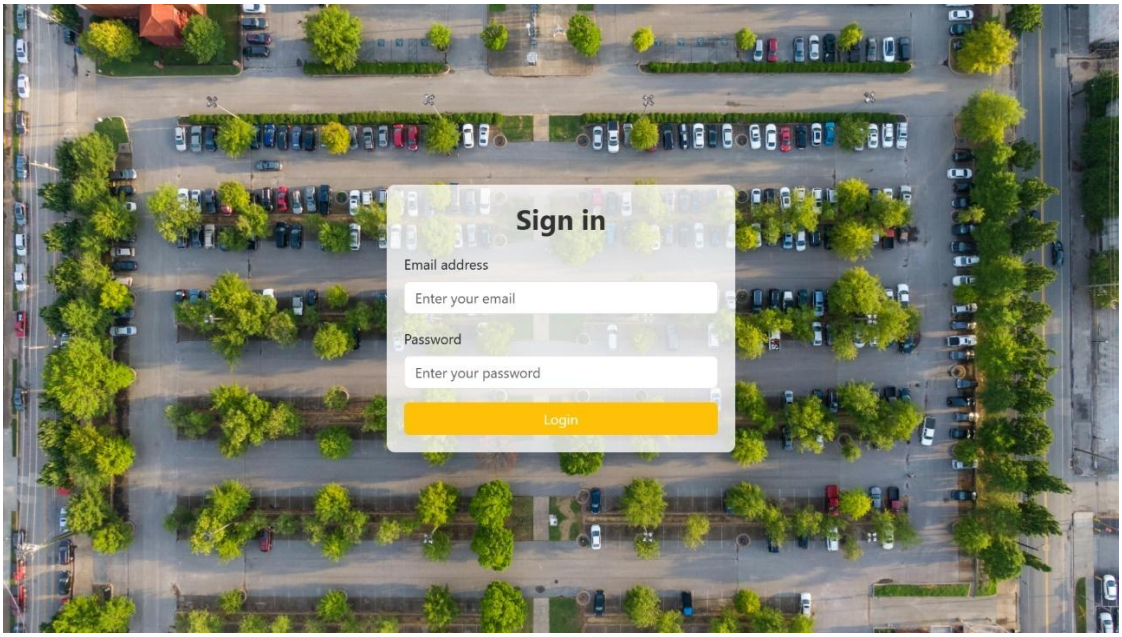
Front Page



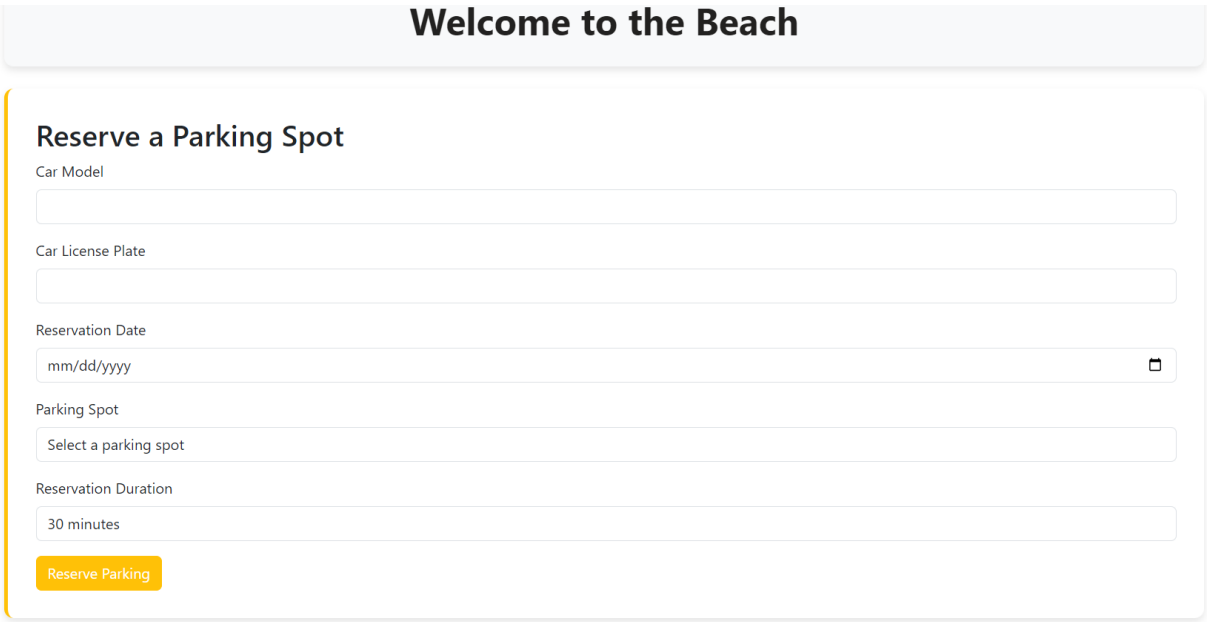
Sign Up Screen



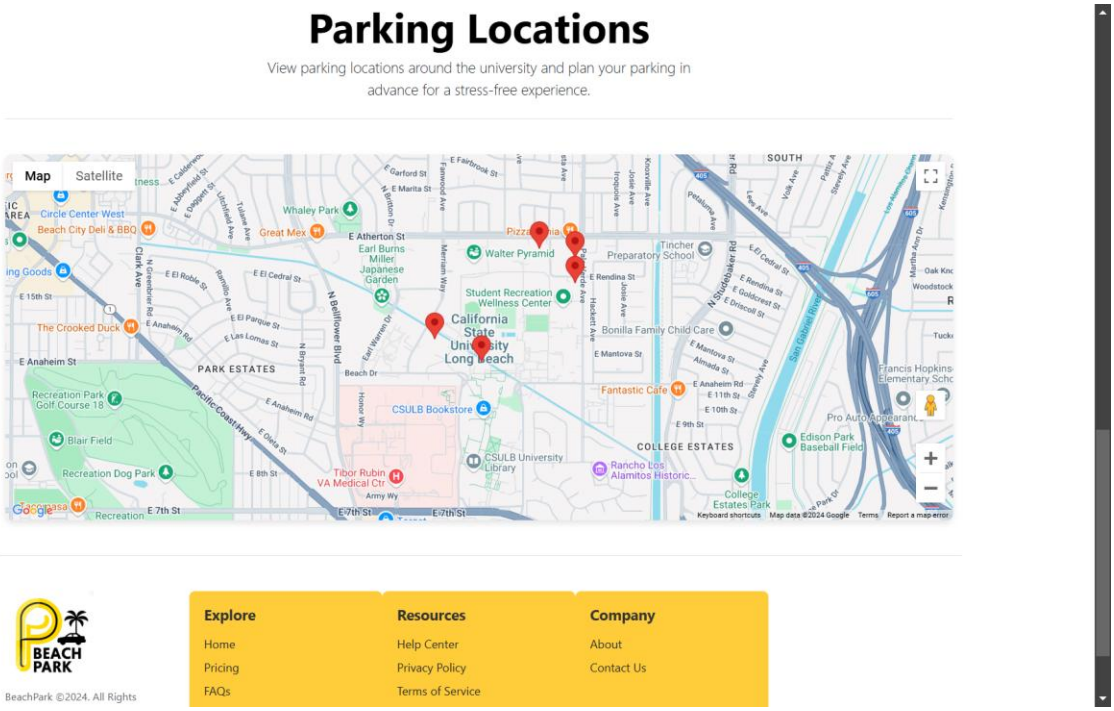
Sign In



Reservation Page



Parking Map



Software Revision

We gave unique names to variables and functions to make it understandable and avoided magic numbers. Comments were added to explain parts of the code to make it easier to understand. Unused or extra code was removed to keep everything clean.

The system was able to have communication done smoothly from the microcontrollers to the Raspberry Pi. Data was being requested each 1000 mus to ensure that we can get the most accurate and fast data possible.

The system updated parking spot information quickly. It managed multiple ultrasonic sensors at the same time without slowing down or getting confused via timer peripherals.

Data moving between the system's parts should be safe and secure. Inputs to the web server should be checked to stop harmful data from getting in. Only authorized users should be able to access Firebase and AWS.

We added more sensors to expand parking spots availability. Each zone consisted of 3 sensors. In addition, we took into consideration how this would work in the real world and integrated UART-to-Ethernet for long distance communication.

Miniature Model

A Winter-themed miniature is used to simulate the area of the parking lots. Our model contains 2 parking zones, with 3 spots each (6 total). The 6 sensors are placed near the left and right sides (3 on each side facing inwards). An orange pyramid is the centerpiece of this model. The borders of the model are filled with trees, bushes, snow, and leaves. A BeachPark Sign is draped across the entrance.

Dimensions

Board: 2.5-ft x 1.67-ft

3D-printed-Pyramid:

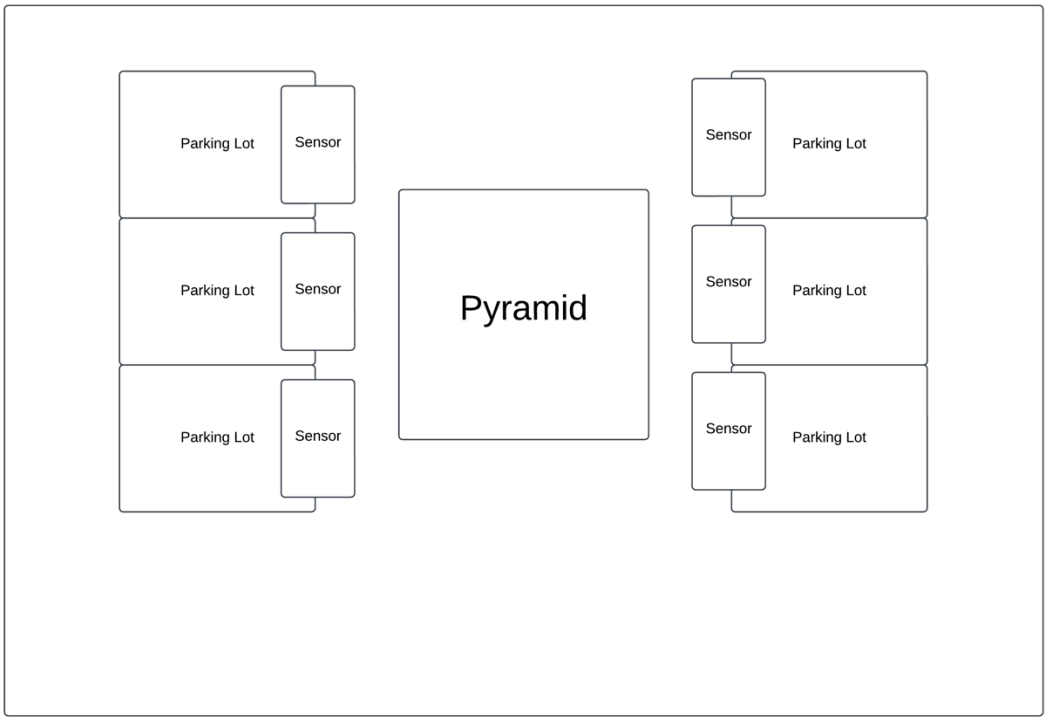
- Base: 4.45-inches x 4.45-inches
- Height: 3.5-inches

Miniature Images:

Front View from Entrance

Above View

Miniature Layout



References

https://www.ti.com/product/TM4C123GH6PM

https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-its.2018.5207

https://www.laexpresspark.org/technology/apps/

https://www.parksmarter.com

https://www.ripublication.com/ijaer18/ijaerv13n8_25.pdf

https://ieeexplore.ieee.org/document/9316049

https://publications.eai.eu/index.php/inis/article/view/4294/2660

https://www.seeedstudio.com/blog/2024/04/09/smart-parking-for-smarter-cities-building-an-end-to-end-data-pipeline-with-raspberry-pi-hmis-aws-tensorflow-and-n3uron/

https://www-sciencedirect-com.csulb.idm.oclc.org/science/article/pii/S2542660521000317

https://go-gale-com.csulb.idm.oclc.org/ps/i.do?p=OVIC&u=long89855&id=GALE%7CA636198109&v=2.1&it=r&aty=ip

https://www.proquest.com/docview/2557076228?pq-origsite=primo&sourcetype=Scholarly%20Journals

https://iopscience.iop.org/article/10.1088/1742-6596/1015/3/032189/meta

https://oscilloscopesguide.com/how-to-use-an-oscilloscope-to-measure-and-analyze-noise/

https://3dsolved.com/create-models-for-3d-printing/

https://www.arrow.com/en/research-and-events/articles/using-capacitors-to-filter-electrical-noise

https://colorhunt.co/palette/f5f7f8fcde70e8b86d185519

https://www.canva.com/

https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

https://www.appbrewery.co/p/web-development-course-resources

https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css

https://getbootstrap.com/docs/5.3/utilities/api/

https://console.cloud.google.com/marketplace/product/google/maps-backend.googleapis.com?pli=1&inv=1&invt=AbkhXw&project=dynamic-parity-445109-a6