Back to previous view

[PAR-125] [item POS] Parking - Intercom System Created: 10/08/24 Updated: 10/15/24

Your computer's time zone does not appear to match your JIRA time zone preference of (GMT+00:00) UTC.
Update your JIRA preference

Status:	New
Project:	Cubework Parking
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Major
Reporter:	Bassel Matta	**Assignee:**	Unassigned
Resolution:	Unresolved	**Votes:**	0
Labels:	Parking		
Remaining Estimate:	Not Specified		
Time Spent:	Not Specified		
Original Estimate:	Not Specified		

Attachments:	image-2024-10-07-16-48-15-810.png
Dev Hours:	0
Story Points:	0
Test Hours:	0
Test Points:	0

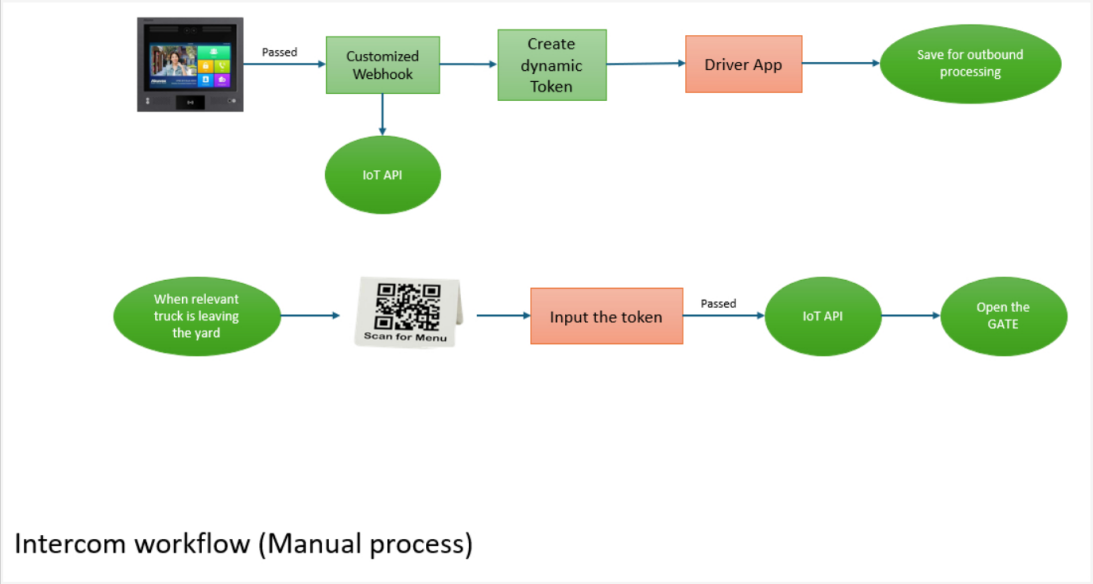
Description

Overview:

In this parking lot access control system, Kevin Chang provides a 4-digit access code to truck drivers, which they enter into an intercom to gain access to the lot. The system will decrypt this 4-digit code into a main internal code stored in the database and validate it. If the code matches, the gate opens, and the system starts recording the truck's entry time, associating it with the specific truck (based on the code and other metadata like time). When the driver enters the lot, they are provided with a new 4-digit exit code. The exit process involves scanning a QR code, entering the exit code in the app, and opening the outbound gate.

This will act as an emercency fast way to get the driver in the yard as quickly as possible to avoid any traffic by the gate in case the camera is not working.

This will not have any relationship to the camera or to the parking app. It will not be connected to the parking app to send the check-in time to the app. There is no need for that for PHASE one.



Intercom workflow (Manual process)

Key Features and Flow:

1. **Inbound Gate Access**:

- Cubework provides a 4-digit code to the truck driver.
- The driver enters this code into the intercom system.
- The system receives the code and verify it is valid.
- If code is invalid, don't open the gate.
- If the code is valid, do the following:
- Open the gate and Call the parking app API to save the 4 digit code and store it in the parking app so the driver can input it when they are exiting the yard to make outbound process.
- If the code matches, the inbound gate opens, and the system logs the truck's entry with the following details: **(PHASE 2)**
  - Truck identifier/LP # (e.g., Truck ABC).
  - Code used (e.g., 1234).
  - Entry time (e.g., 12:03 PM on Tuesday, 10/7/24).

1. **Recording Entry and Linking Truck: (PHASE 2)**

- The system links the truck LP# (based on the code provided to the driver) and logs it along with the entry time.
- This record helps track the truck's history and provides insights for future reference.
- In the backend system in cubework seller dashboard they can assign the code 1022 to truck with LP# HGTHH125 so when the code 1022 is entered in the intercom at 12:00 PM, we know that truck with LP# HGTHH125 has entered the gate.

1. **Exit Process**:

- When a truck enters, a unique exit code is generated for that truck.
- **On exiting, the driver**:
  - Scans a QR code at the exit.
  - Logs into the system using phone number/email.
  - Clicks "Enter Exit Code" and inputs the new 4-digit code.

- If the exit code matches, the outbound gate opens, and the system logs the truck's exit time.

BRD (Business Requirements Document)

1. Objective:

Develop a gate access system that allows drivers to enter and exit parking lots using 4-digit codes provided by Kevin Chang, without camera assistance. This system will track trucks' inbound and outbound movements based on the codes entered, and record both entry and exit times.

This is used in case of an emergency when the camera is off or when app is down.

2. Stakeholders:

- **Kevin Chang**: Provides the 4-digit codes to drivers.
- **Truck Drivers**: Use the system to enter and exit the parking lot.
- **System Admin**: Oversees the parking lot system and manages data.

3. Functional Requirements:

- **Intercom Code Entry System**:
  - Allow the driver to input a 4-digit code via the intercom at the gate.
  - Decrypt the code and match it with the main code stored in the system.
  - Open the gate if the code is valid.
- **Truck Entry Logging**:
  - Log truck ID, code, and entry time.
  - Associate the truck ID with the 4-digit code used for entry.
- **QR Code Scanning for Exit**:
  - Generate an exit code when the truck enters.
  - At the exit, the driver scans a QR code and inputs the exit code in the app.
  - Open the gate if the exit code is valid and record exit time.

4. Non-Functional Requirements:

- **Performance**: System must respond to code inputs and gate operations within 3 seconds.
- **Reliability**: The system should ensure that all entry and exit logs are accurately recorded.
- **Security**: All codes must be encrypted during transmission and securely stored.

5. Assumptions:

- There is no camera functionality to support visual verification.
- All codes are generated, managed, and provided by Kevin Chang.

6. Constraints:

- Real-time validation is required.
- QR scanning needs to be operational for all exiting trucks.

Acceptance Criteria:

1. The system should allow the driver to enter a 4-digit code into the intercom.
2. Upon code entry, the system decrypts the code to match the main code in the database.
3. If the code matches, the gate should open and the truck's entry time should be logged.
4. The system should link the code used with the truck ID and log the entry time.
5. A new 4-digit exit code should be generated and given to the driver for outbound access.
6. The driver should scan a QR code on exit, log into the app, and input the exit code to open the gate.
7. The exit time should be recorded upon successful code entry.

Comments

Comment by Bassel Matta [10/08/24]

Detailed Workflow Breakdown:

1. **Intercom Input (Driver Check-In)**:
   - **Driver Input**: The truck driver approaches the gate and enters the **4-digit code** provided by Kevin Chang into the **Intercom**.
- **Customized Webhook**: Once the driver submits the code, the intercom triggers a **customized webhook** to send this code to the parking app's backend through Marcos' API. The webhook contains the 4-digit code and metadata (like truck ID, entry time).
- **IoT API Validation**: The parking app uses the **IoT API** to validate the 4-digit code. This API checks if the code is valid by decrypting it and matching it against the stored data in the **local MySQL database**.
- If the code is valid:
  - The **IoT API** sends a response to open the gate.
  - A **dynamic token** (exit code) is generated for the driver, which will be required during exit.
  - This dynamic exit code is created and sent to the driver's app, which also stores the data for **outbound processing** (exit).

1. **Truck Exit Process**:
   - **QR Code Scan**: When the truck is ready to leave the yard, the driver **scans the QR code** near the exit gate. This QR code prompts the driver to input the dynamic token (exit code) they were provided when entering.
   - **Token Input and Validation**: The driver enters the dynamic exit code, which is sent back through the **IoT API** for validation. The system compares the exit code with the one stored for that truck's session in the local MySQL database.
   - **Open Gate**: If the token is valid, the **IoT API** triggers the gate to open, logging the exit time, and closing the truck's parking session.

Workflow Steps Aligned with the Image:

1. **Check-In (Upper Flow)**:
- **Intercom**: The driver enters the 4-digit code.
- **Customized Webhook**: The webhook sends the code to the backend parking system.
- **IoT API**: The IoT API decrypts and validates the code against the local MySQL database.
- **Create Dynamic Token**: If successful, the parking app generates a unique **dynamic exit code** for the truck and provides it to the driver via the app.
- **Driver App**: The driver app receives and stores the exit code for later use.
- **Save for Outbound Processing**: The system saves the entry information and prepares the data for later exit processing.

1. **Exit (Lower Flow)**:
- **QR Code Scan**: The truck is leaving the yard, and the driver scans the QR code at the exit gate.

- **Input Token**: The driver inputs the exit code provided at entry into the app.
- **IoT API**: The system validates the exit code via the IoT API by checking the local MySQL database.
- **Gate Open**: If the exit code is valid, the IoT API instructs the system to open the gate, logging the truck's exit.

Local MySQL Database and API Interaction:

- The **local MySQL database** is a copy of the main system database, which is typically hosted in China. It ensures that even if the main database is unreachable due to internet issues, the parking app can continue functioning in the USA.
- **Data Storage and Syncing**:
  - **Entry and Exit Data**: Both the entry 4-digit codes and exit tokens are stored locally in the MySQL instance. The system logs truck entry/exit times, codes used, and other relevant session information.
  - **Sync to Main System**: Once internet connectivity is restored, data stored locally is synced back to the main database to maintain consistency.

Next Steps for Development:

- **Webhook and IoT API Development**:
  - Implement webhook and API endpoints in the parking app to receive and process the 4-digit entry codes.
  - Create functions to generate and store dynamic exit codes and tie them to specific parking sessions.
- **MySQL Database**:
  - Set up a local MySQL instance to mirror the main database. Ensure mechanisms for syncing when connectivity is available.
- **Security and Encryption**:
  - Ensure data is encrypted during transmission (e.g., 4-digit codes, dynamic tokens) between the intercom, IoT API, and parking app.

Comment by Bassel Matta [10/08/24]

How do we track trucks? (PHASE 2)

To effectively track trucks in the system, we'll employ a combination of identifiers, linked data, and timestamps at various stages of the entry and exit process. The goal is to ensure we have a detailed record of each truck's activity, from the moment they approach the gate to when they leave the lot.

Truck Tracking Process:

1. Truck Identification Upon Entry

- Each truck will be identified and tracked primarily through **one or more key identifiers**, which will allow the system to uniquely recognize and log each truck's activity.
- The primary truck identifiers are:
  - **License Plate Number**: This will be captured manually (since there's no camera for automatic recognition).

2. Code-Based Truck Tracking

- Upon entry, Kevin Chang will provide a **unique 4-digit code** to the truck driver.
  - This code is linked to a specific truck in the system database.
  - The system logs this code as tied to that particular truck for tracking purposes.
- When the driver enters the 4-digit code into the intercom, the system performs the following actions:
  - **Decrypts the code** and matches it to the main code in the system for validation.
  - Associates the **entry time** with the truck ID or license plate.
  - Logs the truck's entry details in the system (time, date, truck ID).

3. Entry Time and Logging

- Once the system verifies the 4-digit code and grants access:
  - It records the **exact time of entry** and associates it with the truck.
  - The **truck's ID (license plate or fleet ID)**, the **code used**, and the **entry time** are logged in the system database.
  - This log entry is critical for later tracking the truck's duration in the lot and for matching the truck with its assigned exit code.

4. Exit Code Generation for Tracking

- At the time of entry, the system will generate a **unique 4-digit exit code** that is also tied to the truck's ID.
- This exit code will be linked to the truck's entire parking session, allowing the system to track and manage when and how the truck leaves the lot.
- The system will store the **truck ID**, **entry code**, and **exit code** together, creating a comprehensive log for tracking.

5. QR Code-Based Exit Process

- When the truck driver is ready to exit:
  - They will **scan the QR code** at the exit gate using a mobile app or a provided scanner.
  - This action prompts the system to **authenticate** the truck's session and verify the truck's current status (i.e., has the truck entered? Has the exit code been issued?).
- The driver then inputs the **4-digit exit code**, which is again matched to the truck's identifier in the system.
- Upon successful validation, the gate opens and the **exit time** is recorded.

6. Exit Time and Duration Logging

- When the truck successfully exits:
  - The system records the **exit time**.
  - The system calculates the **total duration** of the truck's stay in the parking lot (entry time to exit time).
  - This information, along with the truck's **ID**, **entry code**, and **exit code**, is logged for future tracking and reporting.

7. Tracking Across Multiple Visits

- Every time a truck enters the lot, a new **entry code** and corresponding **exit code** are generated, even if the same truck makes multiple visits.
- This creates a **visit history** for each truck, allowing the system to track not only individual parking sessions but also aggregate data over time (e.g., how many times Truck ABC has visited, typical duration of stay, busiest times, etc.).

How the System Tracks Trucks Over Time:

1. **Unique Identifier**:
   - Each truck is tracked using its **license plate number** or another unique identifier (e.g., fleet ID).
   - The **4-digit codes** (entry and exit) serve as session-based identifiers linked to that specific truck.
2. **Comprehensive Logs**:
   - For every entry and exit event, the system logs the following data:
     - **Truck ID** (license plate, fleet ID).
     - **Entry code** and corresponding **exit code**.
     - **Entry time** and **exit time**.
     - **Parking session duration** (calculated upon exit).
3. **Historical Data for Analysis**:
   - All data is stored in a database, which provides an easy way to track the truck's movement across multiple visits.
   - The system can generate reports on:
     - **Truck visit frequency**.
     - **Average parking duration**.
     - **Peak entry and exit times**.

Data Flow Example

Data Flow Example:

Let's take an example scenario:

- **Truck ABC** arrives at the gate on 10/7/24 at 12:03 PM.
  - Kevin Chang gives the driver a **4-digit code (e.g., 1234)**.
  - The driver enters **1234** into the intercom, and the system decrypts it and grants access.
  - The system logs **Truck ABC** entering at **12:03 PM**, using code **1234**.
  - Upon entry, the system generates an **exit code (e.g., 5678)** for Truck ABC.
- When Truck ABC leaves at 3:00 PM:
  - The driver scans the QR code at the exit and inputs the **exit code (5678)**.
  - The system verifies the code, logs the **exit time (3:00 PM)**, and calculates the total stay duration as **2 hours 57 minutes**.

Handling Edge Cases:

1. **Mismatched Truck and Code**:
   - If a truck attempts to use a code not assigned to it, the system will deny access and log the attempt as a **failed validation**.
2. **Forgotten or Lost Codes**:
   - If a driver forgets their entry or exit code, they can retrieve the code by verifying their **truck ID** (license plate) in the app, or a manual override by a system administrator.

Comment by Bassel Matta [10/08/24]

APIs Integrations:

To facilitate the integration between the intercom system (using Marcos' API) and the parking app, we need to establish an API-driven workflow that allows the intercom to send the 4-digit access codes to the parking app, which will then store and process the data in a local MySQL instance. This local database is a copy of the main system database (located in China) and ensures the system functions even during internet outages.

Key Integration Components:

1. Marcos API (Intercom API):

- The intercom system, managed by Marcos' API, is responsible for receiving the 4-digit codes entered by the truck drivers.
- The intercom then communicates with the parking app via API calls to send these codes for validation, processing, and storage.
- This API should provide:
  - **Code Entry**: A mechanism to post or send the 4-digit code to the parking system.
  - **Driver Info (if applicable)**: Driver metadata, such as truck ID, timestamp of the entry, and code entered.
  - **Real-time Updates**: The API should support real-time or near-real-time communication with the parking system for prompt action (gate opening, logging, etc.).

2. Parking App API:

- The parking app must expose endpoints to receive the data from the Marcos API and handle several operations:
  - **Receive Code Data**: An endpoint to receive the 4-digit code and accompanying truck/driver data from the intercom.
  - **Code Validation**: Once the code is received, the parking app decrypts and validates it against the data stored in the MySQL database.
  - **Gate Control Logic**: If the code is valid, trigger a gate-opening mechanism via the local system.
  - **Generate Exit Code**: After a successful entry, the app should generate a unique 4-digit exit code, associate it with the truck, and communicate it back to the driver (via app, intercom, or printed ticket).
  - **Error Handling**: The app should handle cases where the code is invalid or when communication fails (e.g., during internet outages).

3. Local MySQL Database Instance:

- The **local MySQL instance** acts as a fallback, ensuring continuous operation even if the main system (hosted in China) becomes unavailable due to network issues.
- The **local MySQL** database will store:
  - **Truck details** (ID, entry code, timestamp).
  - **Driver information** (if applicable).
  - **Entry and Exit Codes**.
  - **Session details** (entry and exit times, duration).
- **Data Synchronization**:
  - The local instance should be regularly synced with the main database when internet connectivity is restored to maintain consistency across both systems.

Workflow: Integration between Marcos API and the Parking App

1. **Step 1: Driver Enters 4-Digit Code into the Intercom**

- The truck driver enters the **4-digit access code** provided by Kevin Chang into the intercom.
- The intercom system, using Marcos' API, transmits this code to the parking app via a secure API call.

1. **Step 2: API Call to the Parking App**

- Marcos' API sends a **POST request** to the parking app's endpoint, transmitting:
  - **4-digit code**.
  - **Truck identifier** (if captured by the intercom).
  - **Timestamp** of code entry.

1. **Step 3: Parking App Receives the Code**

- The parking app's API receives the incoming code and associated data.
- It decrypts the code (if encryption is used) and performs validation by comparing it with the records stored in the **local MySQL database**.

1. **Step 4: Code Validation and Gate Control**

- If the code is valid:
  - The app triggers a signal to **open the gate**, allowing the truck to enter.
  - The entry event is logged in the **local MySQL database**, recording:
    - **Truck ID**.
    - **Code used**.
    - **Entry timestamp**.
- If the code is **invalid**:
  - The system sends an error response to the intercom, and the driver is notified via the intercom's UI.
  - The failed attempt is logged in the local database for tracking purposes.

1. **Step 5: Exit Code Generation**

- Upon successful entry, the system generates a **new 4-digit exit code**, which is linked to the truck's ID and parking session.
- This exit code is communicated back to the driver, either via the intercom or through the parking app.

1. **Step 6: Exit Tracking**

- When the driver is ready to exit the lot, they:
  - **Scan the QR code** at the exit gate.
  - Enter the **4-digit exit code** provided earlier.
  - The parking app validates this exit code against the session details stored in the local database.
  - If valid, the exit gate is opened, and the exit time is recorded.

Handling Offline Scenarios:

- In case the system loses internet connectivity to the main database in China, the parking app will:
  - **Use the local MySQL database** to validate entry and exit codes, log events, and manage parking sessions.

- Once the connection is restored, the system will **sync data** (entry/exit times, truck details) back to the main database in China to ensure consistency.
- The sync process should handle:
  - **Conflict resolution** if data was updated locally and in the main system simultaneously during an outage.
  - **Periodic syncing** (e.g., every 5 minutes) to minimize discrepancies between the local and main databases.

Security Considerations:

1. **Data Encryption**:

- All communications between the Marcos API and the parking app (and between the app and the MySQL database) should be encrypted using TLS/SSL.
- The 4-digit codes and truck data should also be encrypted in transit and at rest to prevent unauthorized access.

1. **Rate Limiting and Throttling**:

- Ensure that the API can handle high-frequency requests without overloading the system, particularly during peak times.

1. **Backup and Failover**:

- Implement automatic **database backups** for both the local MySQL instance and the main system.
- Use a **failover mechanism** in case the local database becomes unavailable, allowing the system to fall back on the main database (or vice versa) when needed.

Sample API Design for Parking App:

- **POST** `/api/entry`

- **Description**: Receives the 4-digit entry code from Marcos API and validates it.
- **Request Body**:
  json code:
  {
  "code": "1234",
  "truck_id": "ABC123",
  "timestamp": "2024-10-07T12:03:00Z"
  }
- **Response**:
  - 200 OK (if the code is valid and the gate is opened).
  - 401 Unauthorized (if the code is invalid).
- **POST** `/api/exit`

- **Description**: Receives the exit code, validates the session, and opens the gate.
- **Request Body**:
  json code:
  {
  "exit_code": "5678",
  "truck_id": "ABC123",
  "timestamp": "2024-10-07T15:00:00Z"
  }
- **Response**:
  - 200 OK (exit successful).
  - 401 Unauthorized (if exit code is invalid).