Requirements Engineering

FerrySYS

Submitted By: Lydia Sheehan (T00222338)

Computing with Software Development

Date Submitted: 21/04/2024

**Table of Contents**

[1. Introduction/overview 4](#_Toc164099940)

[2. Functional Components 5](#_Toc164099941)

[3. User Requirements 6](#_Toc164099942)

[3.1. FerrySYS will manage Vehicle Types. 6](#_Toc164099943)

[3.2. FerrySYS will manage Departures. 6](#_Toc164099944)

[3.3. FerrySYS will manage Tickets. 6](#_Toc164099945)

[3.4. FerrySYS will perform Administration. 6](#_Toc164099946)

[4. System Requirements 7](#_Toc164099947)

[4.1. System Level Use Case Diagram 7](#_Toc164099948)

[4.2. Manage Vehicle Types 8](#_Toc164099949)

[4.2.1. Create Vehicle Type 8](#_Toc164099950)

[4.2.2. Amend Vehicle Type 11](#_Toc164099951)

[4.2.3. Discontinue Vehicle Type 14](#_Toc164099952)

[4.3. Manage Departures 16](#_Toc164099953)

[4.3.1. Schedule Departures 16](#_Toc164099954)

[4.4. Manage Tickets 20](#_Toc164099955)

[4.4.1. Sell Ticket 20](#_Toc164099956)

[4.5. Perform Administration 25](#_Toc164099957)

[4.5.1. View Yearly Revenue 25](#_Toc164099958)

[4.5.2. Analyse Sales 28](#_Toc164099959)

[5. System Model 31](#_Toc164099960)

[5.1. Level-0 DFD 31](#_Toc164099961)

[5.2. Level-1 DFD 32](#_Toc164099962)

[5.3. Level-2 DFD (Process P1: Manage Vehicle Types) 33](#_Toc164099963)

[5.4. Level-2 DFD (Process P2: Manage Departures) 34](#_Toc164099964)

[5.5. Level-2 DFD (Process P3: Manage Tickets) 35](#_Toc164099965)

[5.6. Level-2 DFD (Process P4: Perform Administration) 36](#_Toc164099966)

[6. Data Model (Class Diagram) 37](#_Toc164099967)

[6.1. Class Diagram 37](#_Toc164099968)

[6.2. Relational Schema 38](#_Toc164099969)

[6.3. Database Schema 38](#_Toc164099970)

[7. Conclusion 40](#_Toc164099971)

# Introduction/overview

The system modelled in the following document is that of a ferry business. A real-world example is the Shannon Ferry, that goes between Tarbert, Co. Kerry and Killimer, Co. Clare.

The hypothetical ferry has one route between two ports, with one ferry travelling between them. The service runs the same timetable 7 days a week.

The customers arrive in their vehicle, and as they embark the ferry, they purchase a ticket from a system ‘manager’. The ticket available for purchase is that of the next upcoming departure, from the selected port. The manager selects the type of vehicle the customer is travelling in. There will be a different charge for each type of vehicle, e.g., car, caravan, minibus, van. The system then generates a ticket. This will be the main transaction of the system.

The system allows managers to edit the vehicle types and their associated costs, the departure timetable, and to perform two administrative tasks. The first will be a review of yearly sales, the second will be an analysis of the popularity of vehicle types, by yearly sales.

# Functional Components

# User Requirements

## FerrySYS will manage Vehicle Types.

* + 1. FerrySYS will create a Vehicle Type.
    2. FerrySYS will amend a Vehicle Type.
    3. FerrySYS will discontinue a Vehicle Type.

## FerrySYS will manage Departures.

* + 1. FerrySYS will schedule Departures.

## FerrySYS will manage Tickets.

* + 1. FerrySYS will sell Tickets.

## FerrySYS will perform Administration.

* + 1. FerrySYS will display Yearly Revenue Information.
    2. FerrySYS will Analyse Sales.

# System Requirements

Of the four top-level modules in FerrySYS, the first is Manage Vehicle Types. This allows the manager to manage the price per each vehicle. The second module, Manage Departures, is focused on managing the schedule of departing ferries. The third module, Tickets, is for selling tickets to customers. The final module Administration allows the manager to view graphically depicted ticket sale statistics.

## System Level Use Case Diagram

The following system level use case diagram illustrates the high-level system requirements.

Manager

Customer

* 1. **Manage** Vehicle Types

The functions in this module allow the manager to manage the price (i.e., How much a customer will be charged depending on the vehicle they’re travelling in) and availability (i.e., Whether a ticket can still be sold for that type of vehicle) of each vehicle type.

### Create Vehicle Type

This function allows the manager to create a ‘Vehicle Type’ – this includes a description e.g., ‘Car’, an identifying two-character code, a corresponding price, and a one-character status to indicate availability.

Manager

<<includes>>

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Create Vehicle Type | |
| **Use Case Id** | 01 | |
| **Priority** | High | |
| **Source** | Manager | |
| **Primary Business Actor** | Manager | |
| **Other Participating Actors** | N/A | |
| **Description** | This function adds a Vehicle Type to the Vehicle Types file. | |
| **Preconditions** | N/A | |
| **Trigger** | N/A | |
| **Expected Scenario** | **Manager** | **FerrySYS** |
|  | **Step 1:** Invokes Create Vehicle Type function.  **Step 5:** Inputs data:   * Code char (2) * Description string (25) * Price number (4,2)   **Step 9:** Confirms. | **Step 2:** Retrieves Vehicle Type data from database:   * Vehicle Code   **Step 3:** Assigns default Status ‘A’ (available) to Vehicle.  **Step 4:** Displays UI for entry of data.  **Step 6:** Validates data:   * Code must be two alphabetic characters, cannot be null, must be unique. * Description cannot be null. * Price must be >= 0, cannot be null. * Code must not already exist in Vehicle Types file.   **Step 7:** Inserts data into Vehicle Types file:   * Code char (2) * Description string (25) * Price number (4,2) * Status char (1)   **Step 8:** Displays confirmation message.  **Step 10:** Resets UI. |
| **Alternate Scenarios** | **Manager** | **FerrySYS** |
| **Manager inputs invalid data** |  | **Step 6:** Invalid data detected in input.  **Step 7:** Displays appropriate error message.  **Step 8:** Return to Step 3. |
| **System unable to write to the Vehicle Types file** | **Step 9:** Confirms okay. | **Step 7:** Fails to write to the Vehicle Types file.  **Step 8:** Displays appropriate error message, e.g., ‘Unable to write to Vehicle Types file. Exiting to Main Menu’.  **Step 10:** Returns to Main Menu UI |
| **Conclusions** | A Vehicle Type was created. | |
| **Post conditions** | Tickets may now be sold for this Vehicle Type. | |
| **Business Rules** | N/A | |
| **Implementation Constraints** | Vehicle Code must be unique. | |

### Amend Vehicle Type

This function allows the manager to amend the price, description, and code for a pre-existing, active Vehicle Type.

<<includes>>

<<extends>>

Manager

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Amend Vehicle Type | |
| **Use Case Id** | 02 | |
| **Priority** | Medium | |
| **Source** | Manager | |
| **Primary Business Actor** | Manager | |
| **Other Participating Actors** | N/A | |
| **Description** | This function amends an already existing Vehicle Type. | |
| **Preconditions** | Vehicle Type must exist in the Vehicle Types file to be amended. | |
| **Trigger** | N/A | |
| **Expected Scenario** | **Manager** | **FerrySYS** |
|  | **Step 1:** Invokes Amend Vehicle Type function.  **Step 4:** Selects Description.  **Step 6:** Amends data:   * Description string (25) * Price number (4,2)   **Step 10:** Confirms. | **Step 2:** Retrieves data from Vehicle Types file: Code, Description and Price of all Vehicle Types with a Status ‘A’.  **Step 3:** Displays Vehicle Type Descriptions in UI for selection.  **Step 5:** Displays the Vehicle Type’s corresponding data in UI for amending.  **Step 7:** Validate data:   * Code must be two characters, cannot be null, must be unique. * Description cannot be null. * Price must be >= 0, cannot be null.   **Step 8:** Updates record in Vehicle Types file:   * Code char (2) * Description string (25) * Price number (4,2)   **Step 9:** Displays confirmation message.  **Step 11:** Resets UI. |
| **Alternate Scenarios** | **Manager** | **FerrySYS** |
| **System unable to read from Vehicle Types file** | **Step 4:** Confirm okay. | **Step 2:** Fails to read from Vehicle Types file.  **Step 3:** Displays appropriate error message.  **Step 5:** Returns to Main Menu. |
| **The Vehicle Types file contains no data** | **Step 4:** Confirm okay. | **Step 2:** Retrieval from the Vehicle Types file contains no data.  **Step 3:** Displays appropriate message.  **Step 5:** Returns to Main Menu. |
| **Manager inputs invalid data** |  | **Step 7:** Invalid data detected in input.  **Step 8:** Displays appropriate error message.  **Step 9:** Return to Step 3. |
| **Conclusions** | A Vehicle Type has been amended. | |
| **Post conditions** | The Vehicle with amended details is now available for sale. | |
| **Business Rules** | A Vehicle Type’s Code cannot be amended.  Only available Vehicle Types can be amended. | |
| **Implementation Constraints** | N/A | |

### Discontinue Vehicle Type

This function allows the manager to discontinue the availability of a Vehicle Type. This means setting its Status to ‘U’ (unavailable), rendering it unavailable for sale.

Manager

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Discontinue Vehicle Type | |
| **Use Case Id** | 03 | |
| **Priority** | Medium | |
| **Source** | Manager | |
| **Primary Business Actor** | Manager | |
| **Other Participating Actors** | N/A | |
| **Description** | This function renders a Vehicle Type unavailable for sale. | |
| **Preconditions** | A Vehicle Type must exist and have a Status ‘A’ (available) to be discontinued. | |
| **Trigger** | N/A | |
| **Expected Scenario** | **Manager** | **FerrySYS** |
|  | **Step 1:** Invokes Discontinue Vehicle function.  **Step 4:** Selects a Vehicle Type to discontinue by Description.  **Step 6:** Confirms ‘Discontinue’.  **Step 10:** Confirms. | **Step 2:** Retrieves data:   * All data from Vehicle Types file with a Status of ‘A’.   **Step 3:** Displays ‘Discontinue Vehicle Type’ UI.  **Step 5:** Displays corresponding data from Vehicle Types file in UI for review:   * Code * Price   **Step 7:** Sets Status to ‘U’ (unavailable).  **Step 8:** Updates Status in Vehicle Types file.  **Step 9:** Displays confirmation message.  **Step 11:** Reset UI. |
| **Alternate Scenarios** | **Actor** | **System** |
| **Vehicles file not found** | **Step 4:** Confirm okay. | **Step 2:** Fails to find Vehicle Types file.  **Step 3:** Displays appropriate error message, e.g., ‘Tickets file not found. Exit to Main Menu’.  **Step 5:** Displays Main Menu UI |
| **The Vehicle Types file contains no data** | **Step 4:** Confirm okay. | **Step 2:** Retrieval from the Vehicle Types file contains no data.  **Step 3:** Displays appropriate message.  **Step 5:** Returns to Main Menu. |
| **Conclusions** | A Vehicle Type has been discontinued. | |
| **Post conditions** | The discontinued Vehicle can no longer be sold. | |
| **Business Rules** | Only available Vehicle Types can be discontinued. | |
| **Implementation Constraints** | N/A | |

* 1. Manage Departures

This module allows the manager to create a timetable of departures.

* + 1. Schedule Departures

This function allows the manager to create a daily timetable of departures by selecting which of the two ports the ferry’s journey will begin, at what time it will begin; as well as the time interval a crossing takes from one port to another. The system will then calculate and assign the values of the departure time (Dep\_Time), the arrival time (Arr\_Time), and the port (Dep\_Port) of departure – and then, populate the Departures file with these values.

The first Dep\_Port will be that of the corresponding Port P\_Name the user selects. In each subsequent record generated, it will alternate between ports.

The first Dep\_Time will take the value of the starting time the manager selects. For each subsequent record in the file, the Dep\_Time will be itself, plus the ‘Journey Length’ the manager selects, plus the 10-minute buffer period for disembarking/embarking.

The Arr\_Time for each record will be the Dep\_Time added to the value selected for ‘Journey Length’.

The function will generate records until either of the values for Dep\_Time or Arr\_Time exceeds a value selected by the manager to ‘End By’.

Each record will have an identifying Dep\_ID, as well as a Dep\_Status for indicating whether a Departure is from an old timetable, i.e., inactive, or whether it’s from the timetable currently in use, i.e., active.

Manager

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Schedule Departures | |
| **Use Case Id** | 04 | |
| **Priority** | High | |
| **Source** | Manager | |
| **Primary Business Actor** | Manager | |
| **Other Participating Actors** | N/A | |
| **Description** | This function populates the Departures file based on input from the manager. | |
| **Preconditions** | The two ports must exist in the Ports reference file. | |
| **Trigger** | N/A | |
| **Expected Scenario** | **Manager** | **FerrySYS** |
|  | **Step 1:** Invokes Schedule Departures function.  **Step 4:** Inputs data:   * Selects a ‘Starting Port’ P\_Name. * Selects a ‘First Departure’ value: String (hh:mm). * Selects a ‘Journey Length’: String (hh:mm). * Selects an ‘End By’ value: String (hh:mm).   **Step 9:** Confirms.  **Step 13:** Confirms. | **Step 2:** Retrieves data:   * Departures with a Dep\_Status of ‘A’ (active) from Departures file. * P\_Code and P\_Name from Ports file.   **Step 3:** Displays ‘Schedule Departures’ UI.  **Step 5:** Validates data:   * ‘First Departure’ must be in a valid time format. * ‘Journey Length’ must be > 00:00 and < 24:00. * ‘End By’ must be greater than the sum of ‘First Departure’ and the ‘Journey Length’.   **Step 6:** Calculates, generates, and assigns values for each new record:   * Generates and assigns a Dep\_ID number (9). * Assigns a Dep\_Port char (1). * Calculates and assigns a Dep\_Time time (hh:mm). * Calculates and assigns an Arr\_Time time (hh:mm). * Assigns the Dep\_Status of ‘A’.   **Step 7:** Displays newly generated Timetable  **Step 8:** Prompts user to confirm  **Step 10:** Updates all entries in Departures file to have a Dep\_Status of ‘I’ (inactive).  **Step 11:** Inserts newly assigned values into Departures file:   * Dep\_ID number (9). * Dep\_Time time(hh:mm). * Arr\_Time time (hh:mm). * Dep\_Port char (1). * Dep\_Status char (1).   **Step 12:** Displays confirmation message.  **Step 14:** Resets UI. |
| **Alternate Scenarios** | **Manager** | **FerrySYS** |
| **Departures file and/ or Ports file not found** | **Step 4:** Confirms. | **Step 2:** Fails to find Departures file and/ or Ports file.  **Step 3:** Displays appropriate error message, e.g., “Departures file not found. Exit to Main Menu”.  **Step 5:** Displays Main Menu UI. |
| **No Departures exist** | **Step 4:** Acknowledges. | **Step 2:** No Departures data found.  **Step 3:** Displays appropriate message.  **Step 5:** Continues to Step 4. |
| **Manager inputs invalid data** | **Step 8:** Returns to Step 5. | **Step 6:** Invalid data detected.  **Step 7:** Displays appropriate error message. |
| **Manager changes mind about Timetable** | **Step 9:** Presses Back  **Step 10:** Returns to Step 5. |  |
| **Conclusions** | The Departures file was populated with a new set of Departure records.  The previous active set of Departures have been set to inactive. | |
| **Post conditions** | Tickets can now be purchased for the created Departures. | |
| **Business Rules** | There will be a buffer of 10 minutes between the arrival of the ferry at a port and its departure, to allow for dis/embarking of vehicles. | |
| **Implementation Constraints** | N/A | |

* 1. Manage Tickets

This module contains one function that sells a Ticket to a customer.

* + 1. Sell Ticket

This function generates a Ticket for the next departure of the ferry for a selected Vehicle Type. The system allows the sale of Tickets for the subsequent Departure from the user-selected port only.

A Ticket consists of a generated identifying T\_Code, the date of departure & sale (T\_Date), the time of the sale (T\_Time), the code associated with the Vehicle Type the customer is driving (V\_Code), the cost of the Ticket (Sale\_Price), and the identifying code of the Departure the Ticket is being sold for (Dep\_ID).

Manager

Customer

Customer

FerrySYS

Invoke Sell Ticket

Retrieve relevant data.

Assign default values to Ticket.

Display UI

Make departure port and vehicle selection.

Assign Ticket values based on selection.

Display Ticket data for confirmation

Confirm okay.

Prompt for payment.

Confirms payment received.

Files found?

[Y]

[N]

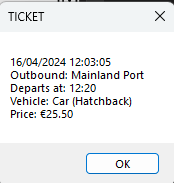
Saves Ticket data to Tickets file.

Prints Ticket

Resets UI

Calculates next upcoming departure

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Sell Ticket | |
| **Use Case Id** | 05 | |
| **Priority** | High | |
| **Source** | Manager | |
| **Primary Business Actor** | Manager | |
| **Other Participating Actors** | Customer | |
| **Description** | This function prints a ticket for the customer based on Vehicle Type for the upcoming Departure and makes a record of the sale in the system. | |
| **Preconditions** | There must be at least 1 entry in each of the following files:   * Vehicle Types file * Departures file.   The two ports must exist in the Ports reference file.  System time and date must be set correctly. | |
| **Trigger** | N/A | |
| **Expected Scenario** | **Manager** | **FerrySYS** |
|  | **Step 1:** Invokes the Sell Ticket function.  **Step 5:** Selects :   * Whether they are on the Mainland or the Island port. * Selects a Vehicle Type by Vehicle Description.   **Step 9:** Confirms.  **Step 11:** Confirms receipt of payment. | **Step 2:** Retrieves data:   * Vehicle Types with a V\_Status ‘A’ from Vehicle Types file * The Port names & codes from the Ports file.   **Step 3:** Assigns values:   * Generates and assigns a number for T\_Code number (9). * Assigns the system date to T\_Date date (DD/MM/YYYY). * Assigns the system time to T\_Time String (hh:mm).   **Step 4:** Displays UI  **Step 6:** Retrieves upcoming departure times from selected Port from the Departures file.  **Step 7:** Assigns values:   * The DepID from the next upcoming Departure from the user selected Port. * The Vehicle Code from the selected Vehicle Type is assigned to the Ticket’s V\_Code char (2). * The Price from the selected Vehicle Type is assigned to the Ticket’s Sale\_Price number (4,2).   **Step 8:** Displays Ticket information for review in UI:   * T\_Date * Dep\_Time of the Departure. * The P\_Name in the Ports file that corresponds to the Dep\_Port of the Dep\_ID. * The V\_Description of the Vehicle Type selected. * Sale\_Price.   **Step 10:** Displays UI prompt for payment.  **Step 12:** Inserts data into Tickets file:   * T\_Code number (9). * T\_Date date (DD/MM/YYYY). * T\_Time time (hh:mm). * V\_Code char (2). * Sale\_Price number (4,2). * Dep\_ID number (9).   **Step 12:** Prints physical ticket.  **Step 13:** Resets UI. |
| **Alternate Scenarios** | **Manager** | **System** |
| **Vehicle Types and/or Departures file not found** |  | **Step 2:** Cannot locate Vehicle Types and/or Departures file.  **Step 3:** Displays appropriate error message, e.g., “Vehicle Types file not found. Exit to Main Menu”  **Step 4:** Returns to Main Menu UI. |
| **Financial Transaction is Unsuccessful** | **Step 11:** Cancels transaction | **Step 12:** Returns to Step 4. |
| **Conclusions** | A Ticket has been generated, sold, and printed. | |
| **Post conditions** | A record of the sold Ticket can now be use in Administrative functions. | |
| **Business Rules** | A Ticket can only be sold for the subsequent Departure only. | |
| **Implementation Constraints** | N/A | |

  
*Fig. 4.6.1.0 The Ticket*

## Perform Administration

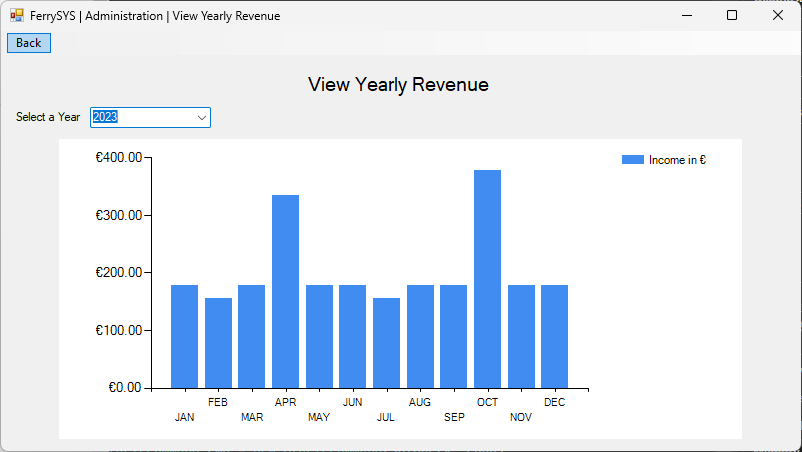
This module allows the manager to perform administrative tasks. This includes viewing the yearly revenue, and viewing the number of tickets sold by vehicle type.

### View Yearly Revenue

This function allows management to view yearly revenue in a graphical format.

Manager

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | View Yearly Revenue | |
| **Use Case Id** | 6 | |
| **Priority** | Low | |
| **Source** | Manager | |
| **Primary Business Actor** | Manager | |
| **Other Participating Actors** | N/A | |
| **Description** | This function produces a graphical representation of the yearly revenue. | |
| **Preconditions** | System must have been in use for a year or more. | |
| **Trigger** | N/A | |
| **Expected Scenario** | **Manager** | **FerrySYS** |
|  | **Step 1:** Invokes the View Yearly Revenue function.  **Step 4:** Selects a year.  **Step 7:** Presses ‘Exit’. | **Step 2:** Retrieves data from the Tickets file: distinct years in T\_Date  **Step 3:** Displays ‘Select a Year’ UI.  **Step 5:** Calculates the sum of Sale\_Price per month from all Tickets with a T\_Date within selected year.  **Step 6:** Displays information graphically in UI.  **Step 8:** Return to ‘Main Menu’ UI |
| **Alternate Scenarios** | **Manager** | **FerrySYS** |
| **Tickets file not found / unable to connect to database** | **Step 4:** Confirms okay. | **Step 2:** Fails to find Tickets file.  **Step 3:** Displays appropriate error message, ‘Tickets file not found. Exit to Main Menu’.  **Step 5:** Displays Main Menu UI |
| **Manager selects another year** | **Step 7:** Selects another year. | **Step 9:** Return to Step 5. |
| **Conclusions** | One or more graphics have been generated. | |
| **Post conditions** | N/A | |
| **Business Rules** | N/A | |
| **Implementation Constraints** | N/A | |

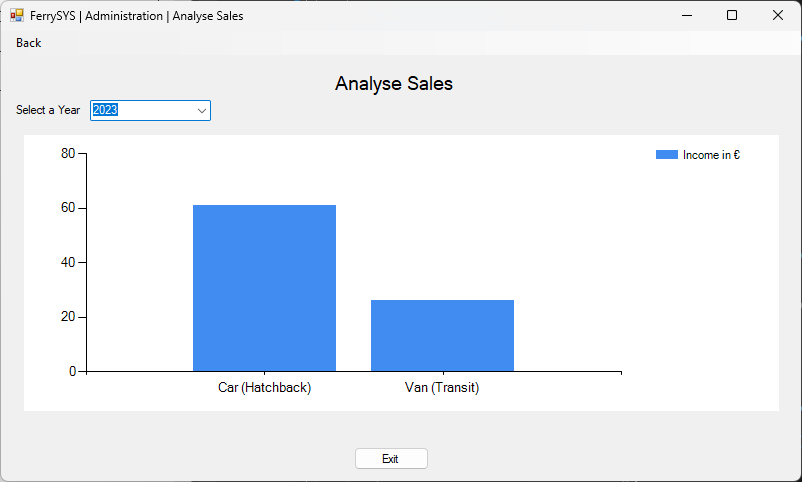
  
*Fig.4.5.1 An example of the graphical yearly revenue*

* + 1. Analyse Sales

This function allows management to view graphical information on the yearly sales sorted by Vehicle Type.

Manager

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Analyse Sales | |
| **Use Case Id** | 7 | |
| **Priority** | Low | |
| **Source** | Manager | |
| **Primary Business Actor** | Manager | |
| **Other Participating Actors** | N/A | |
| **Description** | This function produces a graphical representation of the volume of ticket sales sold per vehicle type. | |
| **Preconditions** | System must have been in use for a year or more. | |
| **Trigger** | N/A | |
| **Expected Scenario** | **Manager** | **FerrySYS** |
|  | **Step 1:** Invokes the Analyse Sales function.  **Step 4:** Selects a year.  **Step 7:** Presses “Exit” | **Step 2:** Retrieves data:   * All data from Tickets file. * V\_Description from the Vehicles file.   **Step 3:** Displays ‘Select a Year’ UI.  **Step 5:** Performs relevant calculations.  **Step 6:** Displays information graphically in UI.  **Step 8:** Returns to Main Menu UI |
| **Alternate Scenarios** | **Actor** | **System** |
| **The system has not been around for more than a year** |  | **Step 2:** Displays appropriate error message.  **Step 3:** Returns to main menu. |
| **Tickets and/or Vehicle\_Types file not found** | **Step 4:** Confirms okay. | **Step 2:** Fails to retrieve Tickets, Vehicle\_Types, Departures and/or Ports file.  **Step 3:** Displays appropriate error message, e.g., ‘Vehicle Types file not found. Exit to Main Menu’.  **Step 5:** Returns to Main Menu UI |
| **Manager selects another year** | **Step 7:** Selects another year. | **Step 8:** Return to Step 5. |
| **Conclusions** | One or more graphics has/have been generated. | |
| **Post conditions** | N/A | |
| **Business Rules** | An analysis will not be performed on a system without a year’s worth of data. | |
| **Implementation Constraints** | N/A | |



*Fig.4.5.2.0 An example of the graphical output from the Vehicle Type analysis*

# System Model

The following dataflow diagrams have been produced for the system:

## Level-0 DFD

FerrySYS

Customer

Ticket details

Ticket

## Level-1 DFD

Ticket details

Ticket details

Vehicle Type details

Vehicle Type Details

Port P\_Name

Departure details

Manage Vehicle Types

P1

Manage Departures

P2

Vehicle Types File

D1

Ports File

D2

Departures File

D3

Tickets File

D4

Manage Tickets

P3

Perform Administration

P4

Vehicle Type details

Port details

Departure Details

Departure Details

Customer

Ticket details

Ticket

Vehicle V\_Description

## Level-2 DFD (Process P1: Manage Vehicle Types)

Create Vehicle Type

P1.1

Discontinue Vehicle Type

P1.3

Amend Vehicle Type

P1.2

D1

Vehicle Types File

Vehicle Type Details

Vehicle Type Details

Vehicle Type Details

Vehicle Type Details

Vehicle Type Details

## Level-2 DFD (Process P2: Manage Departures)

Schedule Departures

P2.1

D3

Departures File

Departure Details

Departure Details

D2

Ports File

Ports Details

* 1. Level-2 DFD (Process P3: Manage Tickets)

Sell Ticket

P3.1

D1

Vehicle Types File

D2

Ports File

D3

Departures File

D4

Tickets File

Customer

Vehicle Type details

Port P\_Name

Departure details

Ticket details

Ticket details

Ticket

* 1. Level-2 DFD (Process P4: Perform Administration)

View Yearly Revenue

P4.1

D1

Vehicle Types File

Analyse Sales

P4.2

D4

Tickets File

Ticket details

Ticket details

Vehicle Type details

# Data Model (Class Diagram)

FerrySYS has four data stores.

The Ports table, the Vehicle Types table, the Departures table, and the Tickets table.

## Class Diagram

+ getPorts(): DataSet

**PORT**

- Code: char  
- Name: String

+ getTimetable(): DataSet  
+ getTimetable(String): DataSet   
+ deactivateDepartures(): bool  
+ getNextID(): int  
+ scheduleDeparture(Departure): bool

**DEPARTURE**

- ID: int  
- Dep\_Time: String  
- Arr\_Time: String  
- Status: char

+ getNextID(): int  
+ saveTicket(Ticket): bool  
+ viewRevenue(String): DataSet  
+ yearsAvailable(): DataSet

**TICKET**

- Code: int  
- TDate: date  
- TTime: String  
- Sale\_Price: decimal

+ validateVehicleInput(String, String, String, List<String>): bool  
+ validateVehicleInput(String, String, String): bool  
+ addVehicleType(): bool  
+ getAvailableVehicleTypes(): DataSet

**VEHICLE TYPE**

- Code: String  
- Description: String  
- Price: decimal  
- Status: char

1.. \*

1..1

departs from

1..1

0.. \*

is purchased for a

1..1

0.. \*

is purchased for a

## Relational Schema

The following is the relational schema for the data requirements for FerrySYS.

Ports (P\_Code, P\_Name).

Vehicle\_Types (V\_Code, V\_Description, Price, V\_Status).

Departures (Dep\_ID, Dep\_Time, Arr\_Time, Dep\_Status, Dep\_Port).

Tickets (T\_Code, T\_Date, T\_Time, Sale\_Price, Dep\_ID, V\_Code).

## Database Schema

The following is a definition of the database to be implemented for FerrySYS.

**Schema:** FerrySYS

**Relation Ports**

P\_Code String (1)

P\_Name String (15) NOT NULL

**Primary Key:** P\_Code

**Relation Vehicle Types**

V\_Code String (2)

V\_Description String (25) NOT NULL

Price Decimal (4) NOT NULL CHECK >= 0

V\_Status String (1) DEFAULT ‘A’ CHECK IN ‘AU’

**Primary Key:** V\_Code

**Relation Departures**

Dep\_ID numeric (9)

Dep\_Time Time NOT NULL

Arr\_Time Time NOT NULL

Dep\_Port String (1)

Dep\_Status String (1) CHECK IN ‘AI’

**Primary Key:** Dep\_ID

**Foreign Key:** Dep\_Port References Ports

**Relation Tickets**

T\_Code numeric (9)

T\_Date Date NOT NULL

T\_Time Time NOT NULL

V\_Code String (2) NOT NULL

Sale\_Price Decimal (4) NOT NULL CHECK >= 0

Dep\_ID numeric (9) NOT NULL

**Primary Key:** T\_Code

**Foreign Keys:** V\_Code References Vehicle Types, Dep\_ID References Departures

# Conclusion

Several tools have been used to explore the functional requirements of the design of the hypothetical FerrySYS system. Use-case narratives were used to detail the seven functions of the system; while data-modelling techniques have been used to detail the data stores the system would need. To describe the interaction of the functions and data stores, data flow diagrams have been used.