Department of Accounting & Information Systems

UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

ACCT/INFO GROUP PROJECT SUBMISSION

CASE STUDY/PROJECT TITLE: INFO213 Course Project Milestone 2: The Terminal Operating System Implementation

Please complete all sections of this sheet, sign the declaration and attach the sheet to your project.

The next panel must be completed by all team members, **including** the agreed proportion of work done on the project. (For example. if all members of a team of four made equal contributions then enter 25% for each team member.)

	Student ID No.	**User ID** e.g. *afg21*	**Student Names:** (**Surname first** & alphabetical order please)	**Proportion %** (Agreed by group)
1)	45548000	ldc40	Creed, Louis	33%
2)	97440534	ljg70	Garton, Lewis	33%
3)	58536578	skh154	Hari, Saahil	33%
4)	....	....	....	……………………….
5)	....	....	....	……………………….

Honesty Declaration

- I declare that this is an original assignment and is entirely my own work.
- Where I have made use of the ideas, words or work of others, I have acknowledged the source in every instance.
- Where I have used any diagrams (including modifications) prepared by others, I have acknowledged the source in every instance.
- I have read and understood the Dishonest or Improper Practices Statement overleaf.
- I am aware of what constitutes cheating, and the penalties for plagiarism and cheating as described in University publications.
- I am aware that the content of this written work may be checked against an electronic database.
- I have supplied the correct word count and have taken no steps to cause disclosure of an incorrect word count for the assessment.

I have read and fully understand the Honesty Declaration above, and hereby certify that this item of work submitted for assessment is entirely the work of the members of the group, in the proportions stated.

Signed . . .

1) Louis Creed	**3)** Saahil Hari	**5)** ....
2) Lewis Garton	**4)**...	

Under the University Regulations, evidence of any of these or other forms of dishonest practice by any student(s) represents grounds for disciplinary action and may result in penalties ranging from denial of credit for the item or work in question, to exclusion from the University.

Dishonest or Improper Practices

It is recognised that students will discuss course work and assignments with others, and such discussion is an important part of the learning process. However, any work presented by a student for credit in a course must be that student's own original work. If students are directed to complete work submitted for credit in groups, the work submitted must be the original work of the group. Work submitted in breach of these requirements or which fails to comply with other instructions contravenes the University's Dishonest Practice and Breach of Instruction Regulations. Such work will either not be marked, and all credit for the work in question forfeited, or the matter will be referred to the University's proctor for investigation and possible referral to the University's Disciplinary Committee.

Penalties which may be imposed in the event of a finding of dishonest or improper practice include loss of credit for a course or an item of assessment and, in serious cases, suspension or expulsion from the University. A record is kept of all instances of dishonest conduct.

Instances of dishonest or improper practice in coursework and assignments include but are not limited to:

- Plagiarism. Plagiarism means the dishonest presentation of work that has been produced by someone else as if it is one's own. Please note that the presentation of someone else's work as one's own, even without dishonest intent, may still constitute poor academic practice, and this may be reflected in the mark awarded. There are academic conventions governing appropriate ways to acknowledge the work or part of the work of another person, including the APA and Harvard citation styles. For further information see the UC Library website, under "Citations and Referencing".
- Submitting for credit in a course without the prior consent of the Course Coordinator for an essay, research paper or any other written work which, although it is the student's own work, is substantially the same as work which has already been (or will be) submitted for credit in another course, whether in the Department of Accounting and Information Systems (ACIS Department) or some other department or academic institution.
- Copying the work of another student. This includes copying the work submitted by another student for credit for a course in the ACIS Department or some other department or academic institution.
- Knowingly allowing another student to copy work which that other student then submits for credit for a course in the ACIS Department.
- Arranging for another person to complete work which is then submitted for credit for a course in the ACIS Department. An example falling in this category is work submitted for credit which has been obtained from a commercial assignment completion service. Care must be taken when using editing services as it is **only** assistance with grammar, punctuation and expression that is permissible and does **not** include the addition or amendment of content.
- Completing work for another student which is then submitted by that other student for credit for a course in the ACIS Department.
- Including made up or fabricated material in work submitted for credit for a course in the ACIS Department.
- Collaborating in the preparation of answers for take home or online tests unless advised otherwise in the take home test instructions.

If you are in doubt about any of the above with respect to a particular course, you should discuss the matter with the lecturer or course co-ordinator concerned.

See also the University Discipline Regulations, Dishonest Practice and Breach of Instructions Regulation, and Academic Integrity Policy – refer to UC Calendar and UC web.

Gate Operations Prototype
INFO213 Project Milestone 2
Monday 4th June 2018

As part of Milestone 2 we implemented the Gate Operations UML. Saahil modelled the UML as part of Milestone 1.

Project Scope
The project scope implements the core functionality of the Gate Operations UML. The Gate Operations subdomain tracks vehicle cargo data as it transits to and from a terminal through terminal gates.

The core objects contained in the TerminalSystem Schema are terminal, cargo, transport company, vehicle and driver. Further objects are built from the core objects to keep a record of relevant Gate Operations. These are cargo damage records, cargo damage tracking system, vehicle visit record and vehicle booking system. Member-by-key dictionaries keep track of the collections of system objects.

An MDI interface allows for the management of core objects from a single main menu. Each core objects can be listed in tabular format and modified according to the particular use case. The use case functionality that can be verified by the GUI is detailed below.

XML Parsers for cargo, transport company, driver and vehicle objects are built into the system. Each parser accept a XML document for a single core object and creates persistent objects within the system.

A basic testing framework is included. The testing framework provides unit tests to verify the functionality of the core objects, exception handlers and the collections which are built from them.

Correctly formatted datasets for each object have been included to allow core domain objects to be imported into the system. Broken datasets have been included to verify exception handling functionality.

Whilst managing multiple gates and terminals was originally planned for the scope of this project, time constraints and project design meant this was omitted. Some of the underlying logic for gate management is included in the project, however this is not used in the GUI, which instead opens a popup notification to inform the user that the feature is not yet implemented.

Functionality
The scope of the prototype provides for the following use cases to be completed:

- Management of cargo
  Cargo objects can be listed in a table and then added, edited or removed. In addition, cargo XML Documents can be imported as persistent objects. Management of cargo is provided through the Management tab in the GUI.

- <u>Management of transport companies</u>
  Transport company objects can be listed it tabular format added, edited or removed. Transport company XML Documents can be imported into the system. Accessed through the Management tab.

  <u>Management of vehicles</u>
  Vehicles sit underneath Transport Company in the containment hierarchy. Vehicle objects can be added and edited. Vehicle use cases are accessed via a submenu of Transport Company in the Management tab.

  <u>Management of drivers</u>
  Drivers also sit underneath Transport Company in the containment hierarchy. Driver objects can be added and edited. Driver use cases are accessed via a submenu of Transport Company in the Management Tab.

- <u>System records of vehicle visits</u>
  Vehicle visits are recorded by the system. Vehicle visit records can be added, edited and deleted. Accessed under the MDI System tab.

- <u>System record of cargo damage records</u>
  Cargo damage records are recorded by the system. A cargo damage record can be added, edited and deleted. Accessed under the System tab.

TerminalSystem

The TerminalSystem Schema implements the classes modelled in the UML Diagram as part of Milestone 1. The classes have been implemented as they were represented in the UML diagram. They include: cargo, cargo damage record, damage tracking system, driver, gate, terminal transport company, vehicle, vehicle visit record and vehicle booking system.

Vehicle and Driver objects reference their Transport Company as myTransportCompany.

The app object contains references to the damageTrackingSystem and visitBookingSystem objects.

The JadeScript class has been included to test the application code during development. It contains various methods that were used to create and delete objects, and parsing methods to test XML functionality.

System Collections

The following member-key dictionaries are implemented in the system to track of important objects:

- CargoByID: tracks the current cargo objects in the system by their ID attribute
- CargoDamageRecordsbyID: tracks the history of cargo damage records at a terminals by ID
- DriversByLicence: tracks vehicle drivers entering/exiting a terminal by their ID
- GateByID: tracks the terminal current gate configuration by ID
- TransportCompaniesbyID: tracks the current transport companies by ID
- VehiclesByPlate: tracks vehicles by licence plate no.
- VisitRecordsByID: tracks the history of vehicle visits by visit ID

XML Parsers

XML Documents are handled by XML Parsers implement using the JadeXMLParser API. The Parser reads relevant document serially and feeds data back to the TerminalSystem.

The following Jade XML Parsers have been implemented. The system deals with each core object as a separate XML document.

- CargoParser: accepts a Cargo XML Document and creates cargo objects
- CompanyParser: accepts a Company XML Document and creates transport company objects
- DriverParser: accepts a Driver XML Document a creates driver objects
- VehicleParser: accepts a Vehicle XML Document and creates vehicles objects.

Test XML document for each core object have been provided.

Exception Handling/Logging Functionality

Cargo exception handler

This handler is defined as a method in the cargo class. It ensures that correct cargo inputs are given to the system. Where an exception arises, the current database transaction is aborted, the exception code is written to the TerminalOperatingSystems_error.log file. The app object is called to display a MSG_OK_Only error message to the user describing the error that occured. The implemented cargo error codes are:

- 1035: String too long
- 64000: Empty cargo description
- 64001: Cargo weight recorded as zero
- 64002: Invalid cargo clearance type
- 64003: Empty cargo source
- 64004: Empty cargo destination.

If the error code is not recognised by the cargo exception handler, the exception is passed back to the next armed handler.

The broken datasets in the XML/Cargo folder can be used to test the cargo exception handlers.

Vehicle Exception Handler

The vehicle exception handler is defined as a method in the vehicle class. It ensures that correct vehicle inputs are given to the system. Where an an exception arises, abort transaction is called to release the locks, the exception is logged to the TerminaSystemerrors_log file and a MSG_OK_ONLY box is display to describe the error made to the user.

The implemented vehicle error codes are:

- 1035: String too long
- 65031: Vehicle make string is empty
- 65032: Vehicle model string is empty
- 65033: Vehicle plate string is empty
- 65034: Invalid year entered (must be newer than 1950)

If the error code is not recognised by the vehicle exception handler, the exception is passed back to the next armed handler.

XML Exceptions Handlers:

Due to the nature of exception handlers in the relevant classes, most exceptions caused by invalid data inside an XML file will be caught by the corresponding class' exception handler.

Unit Testing Framework

The Unit Testing framework has been implemented as a subschema of TerminalView named TerminalUnitTests. Included within are the following unit tests:

- DriverTest
- TransportCompanyTest
- CargoTest

The unit tests in this are simple, and only serve to test very basic functionality, such as if objects are being created correctly. This decision was made due to the exception handling providing sufficient feedback to the user in order to be able to identify problems with data importing.

GUI

The GUI is contained in the TerminalView schema, a subschema of TerminalSystem. An MDI Main Menu contains all of the other defined GUI forms. Tables of each relevant object are displayed before a user has the option to complete a use case with respect to that object.

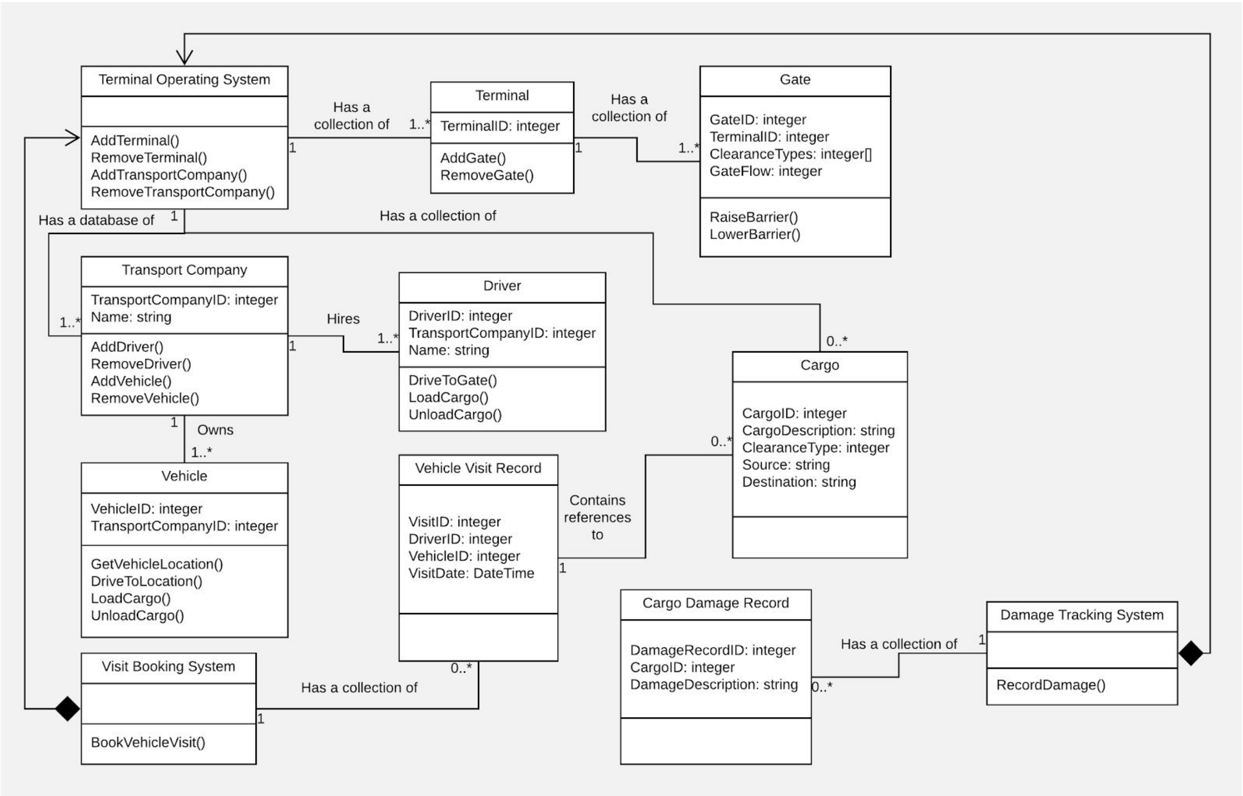
The following tables are included: CargoDamageTable, VehicleVistsTable, CargoTable, DriverTable, TransportCompanyTable, VehicleTable.

The following forms are included: CargoDetails (subforms:CargoAdd and CargoEdit),
Company Details (subforms: Company Add and Company Edit),
DamageRecordDetails (subforms: DamageRecordAdd and DamageRecordEdit),
DriverDetails (subforms: DriverAdd and DriverEdit),
VehicleDetails (subforms: VehicleAdd and VehicleEdit),
VisitDetails (subforms: VisitAdd and VisitEdit).

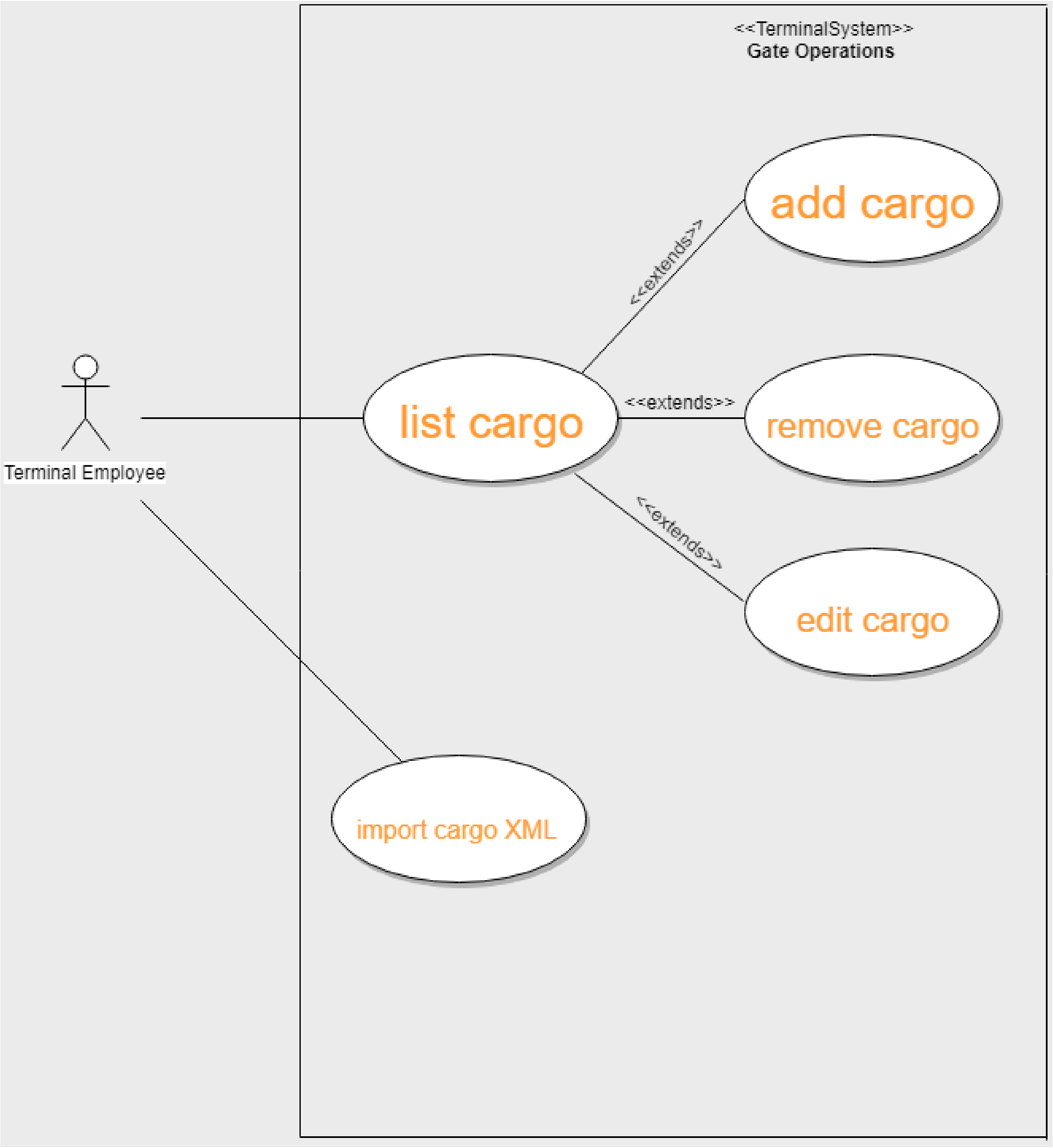
The isDataValid and clearTextBoxes methods ensure that the inputs that are entered into the respective form are correct, and that the textboxes are reset to blank after entry.

Gate Operations UML

The general class layout that we designed the system is shown in the following class diagram. Due to time restraints, design practices and the project scope, certain features were either left out or had their implementation altered from what was originally planned.



Use Case Diagram - Manage cargo


<<TerminalSystem>>
Gate Operations
Terminal Employee
list cargo
<<extends>>
add cargo
<<extends>>
remove cargo
<<extends>>
edit cargo
import cargo XML