SYSTEM DESIGN

cuTPS – Carleton University Textbook Publishing System

Team Do Not Stick In Ear

Andrew MacCuaig
Graeme Jager
David Jatczak
Sina Dee

Table of Contents

Tables

Figures

1. Introduction

1.1. Project Overview

The Carleton University Textbook Publishing System (cuTPS) will provide the means for content providers to bring their written work to students electronically. To implement this main service, a combination of subsystems would be put together. With the intention of keeping any storage of this service on remote hosts to a minimum, a lightweight architecture design was implemented. The purpose of this document will be to decompose the system into subsystems and show how they process information.

1.2. Overview of Document

First step to breaking down the system into smaller parts, the document will focus on subsystem decomposition, design strategies, subsystem services, and class interfaces. The subsystem decomposition will discuss the initial prototype, give further insight into its decomposition, as well as the evolution to the next phase. The design strategy will display how the subsystems will be mapped to components and from components to nodes. It will detail how data storage is handled and what objects are used. The document then shows which subsystems provide services to other subsystems, and finally it will show what classes are used in those services. Each class will display its parameters, functions, and return types.

2. Subsystem Decomposition

2.1. Phase 1 Prototype Decomposition

The older design for cuTPS that came about from Deliverable 2 uses the client-server architecture. Only two subsystems were able to come into existence due to lack of focus on the overall program design and instead efforts were concentrated on providing functional API test drivers. Most of the functionality is grouped by classes which brought about very little in terms of subsystems. Since the only subsystems that exist are the client and server, they are essentially a client-server architecture as shown in Figure 1 and Figure 2.

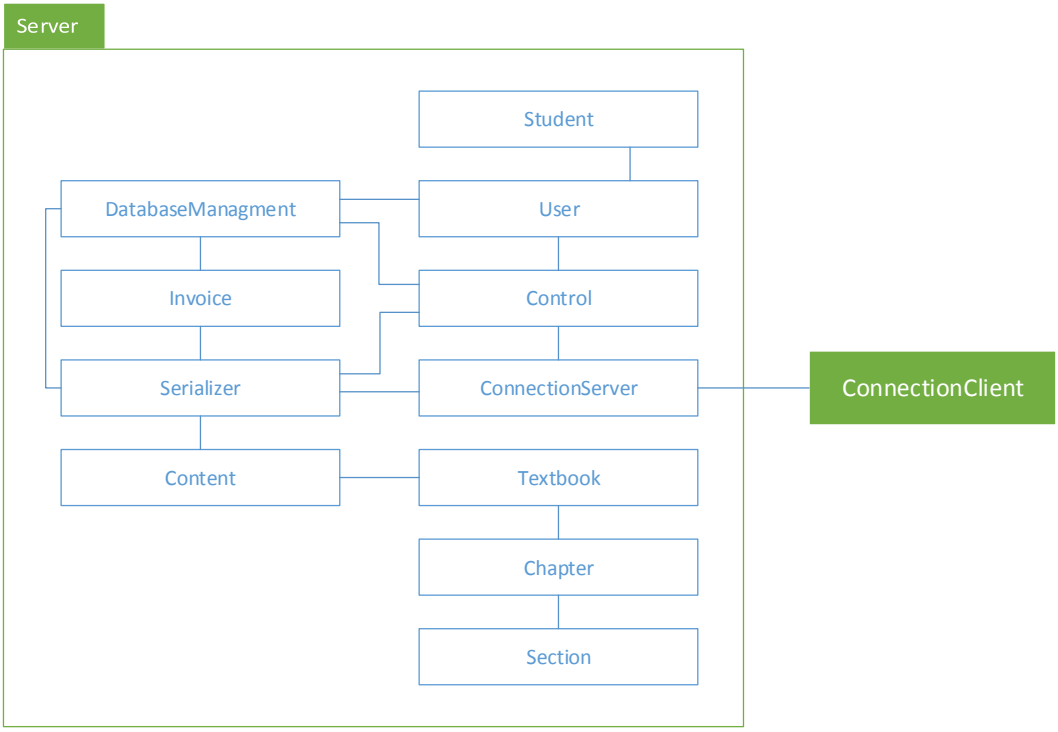

Figure 1 - Phase 1 Server Prototype

Table 1 - Phase 1 Server Prototype Traceability

Identifier	[SS-01]
Name	Server
Traceability	Deliverable 2

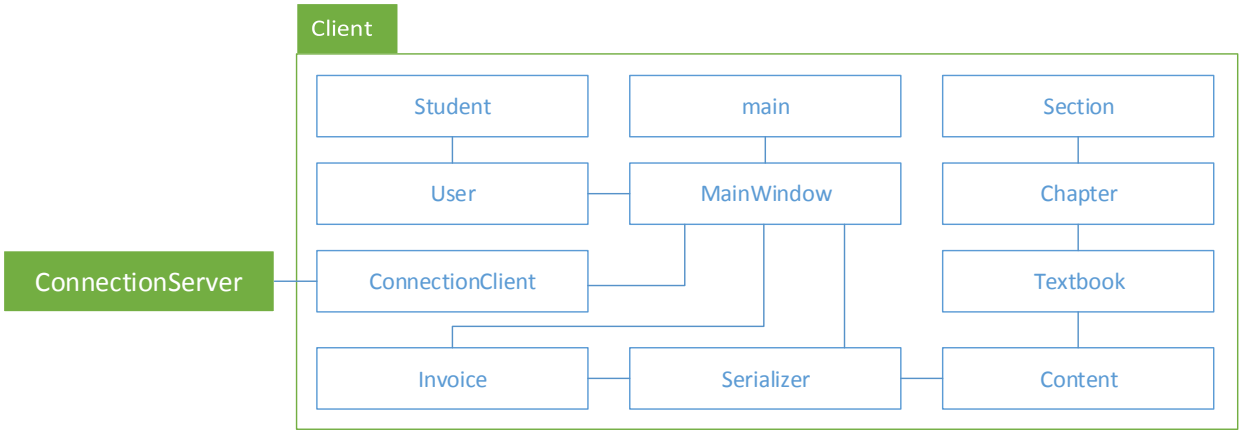

Figure 2 - Phase 1 Client Prototype

Table 2 Phase 1 Client Prototype Traceability

Identifier	[SS-02]
Name	Storage
Traceability	Deliverable 2

2.2. System Decomposition

2.2.1. Overview

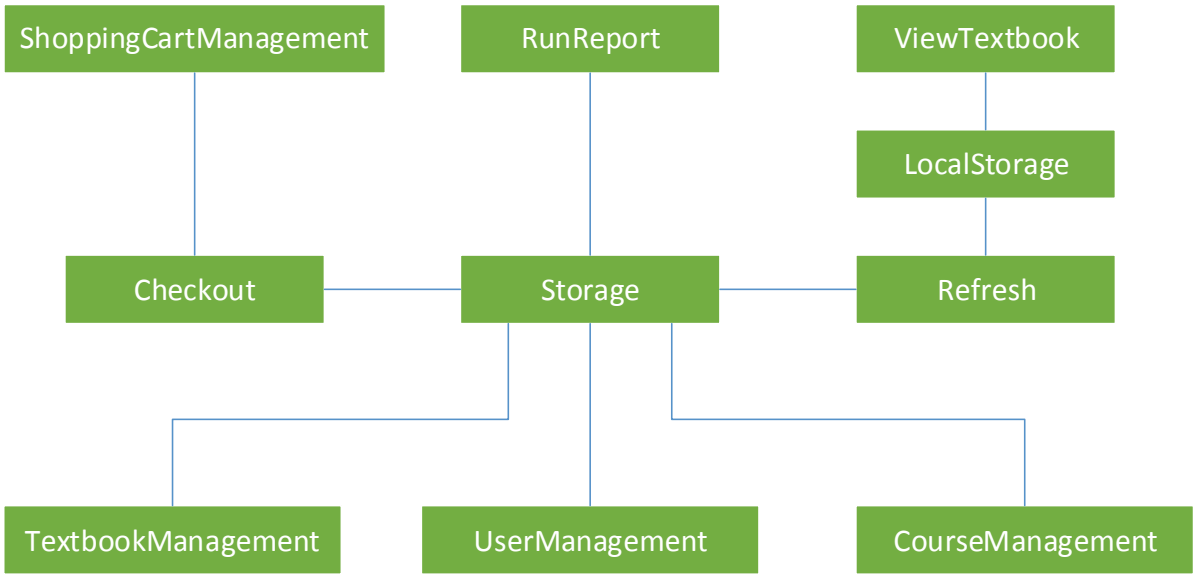

Figure 3 - Subsystem Overview

Table 3 - Subsystem Overview Traceability

Identifier	[SS-03]
Name	Storage
Traceability	Deliverable 2

This figure provides a high-level overview of the subsystems that make up cuTPS. Each subsystem is explained in more detail further on in this document. In this diagram, you can see that the subsystems are separated by the functions that they perform and that each subsystem feeds in to, and receives information from, the Storage subsystem. The pros and cons of this are also explained further on.

2.2.2. Shopping Cart Management

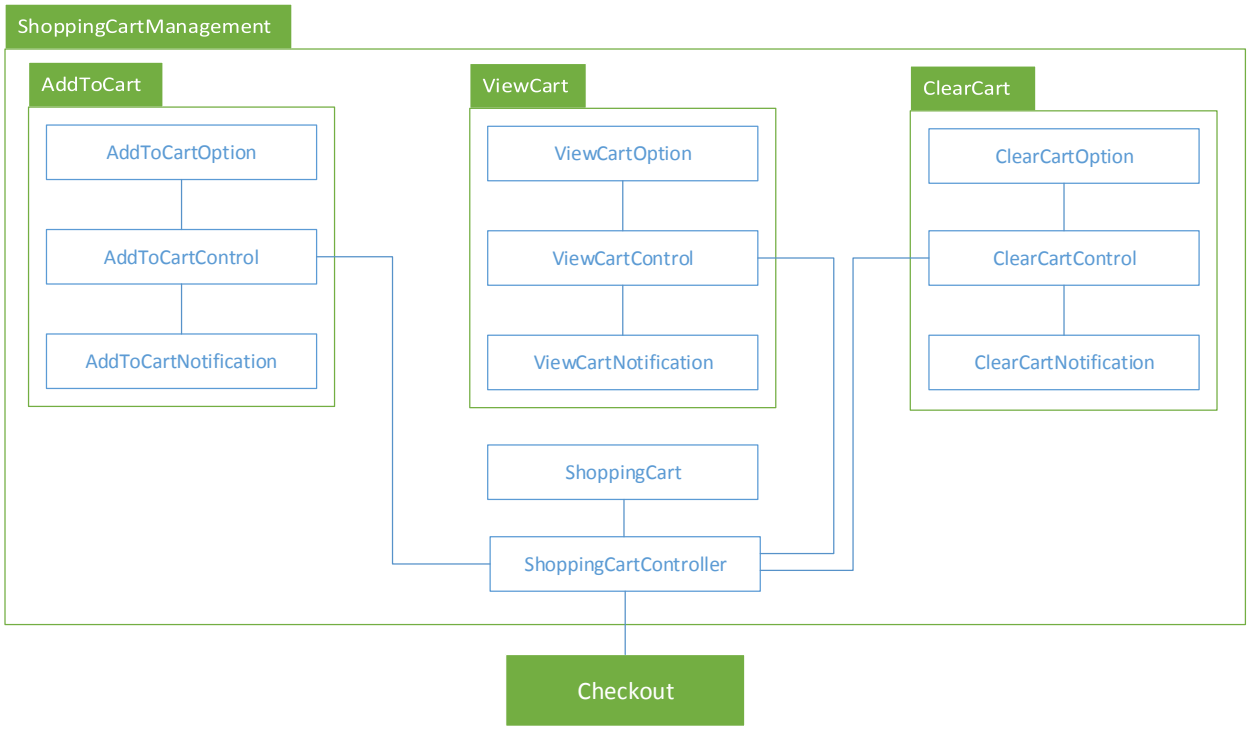

Figure 4 – Shopping Cart Management Subsystem

Table 4 - Shopping Cart Management Subsystem Traceability

Identifier	[SS-04]
Name	ShoppingCartManagement
Traceability	[SD-02], [SD-03] , [SD-05]

The Shopping Cart Management subsystem is broken down into 3 packages each with a clear and succinct function. Each has a boundary object, control object, and notification object. The Shopping Cart Controller object is responsible for the overall management of a user's shopping cart as well as communicating with the Checkout subsystem when a user is ready to complete a purchase.

2.2.3. Checkout

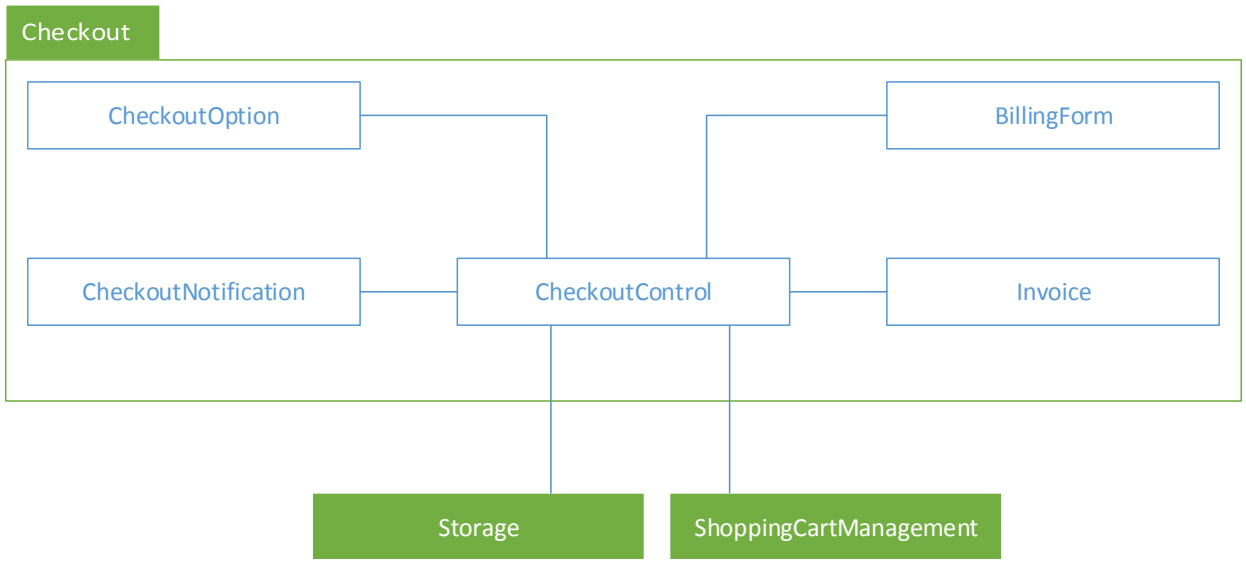

Figure 5 - Checkout Subsystem

Table 5 - Checkout Subsystem Traceability

Identifier	[SS-05]
Name	Checkout
Traceability	[SD-04]

The Checkout subsystem is responsible for handling the entire checkout process once a user has decided to purchase the contents of their shopping cart. This includes retrieving all billing information needed from the user to complete the purchase, creating an Invoice object and sending it to the Storage subsystem. This is all handled and coordinated by the Checkout Control object.

2.2.4. View Textbook

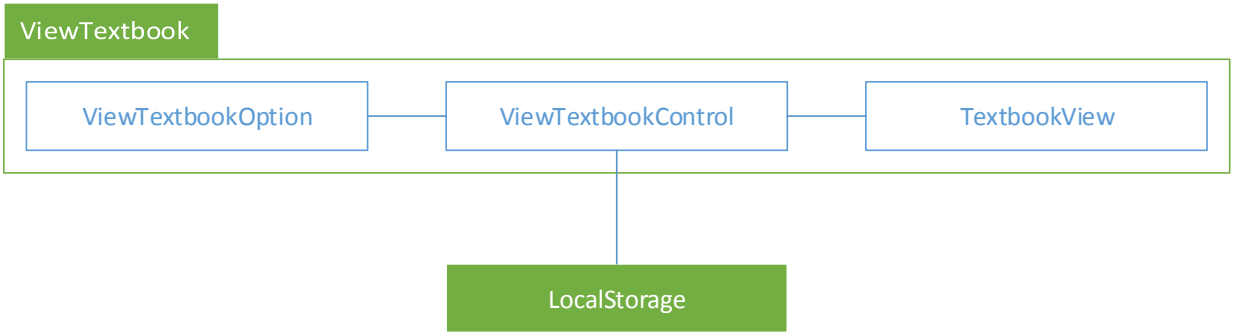

Figure 6 – View Textbook Subsystem

Table 6 - View Textbook Subsystem Traceability

Identifier	[SS-06]
Name	ViewTextbook
Traceability	[F-01]

The View Textbook subsystem is responsible for retrieving all the details about a specific textbook from the Local Storage subsystem and displaying them to the user through the Textbook View object. The View Textbook Option object is responsible for initiating this process but it is the View Textbook Control that coordinates everything.

2.2.5. Local Storage

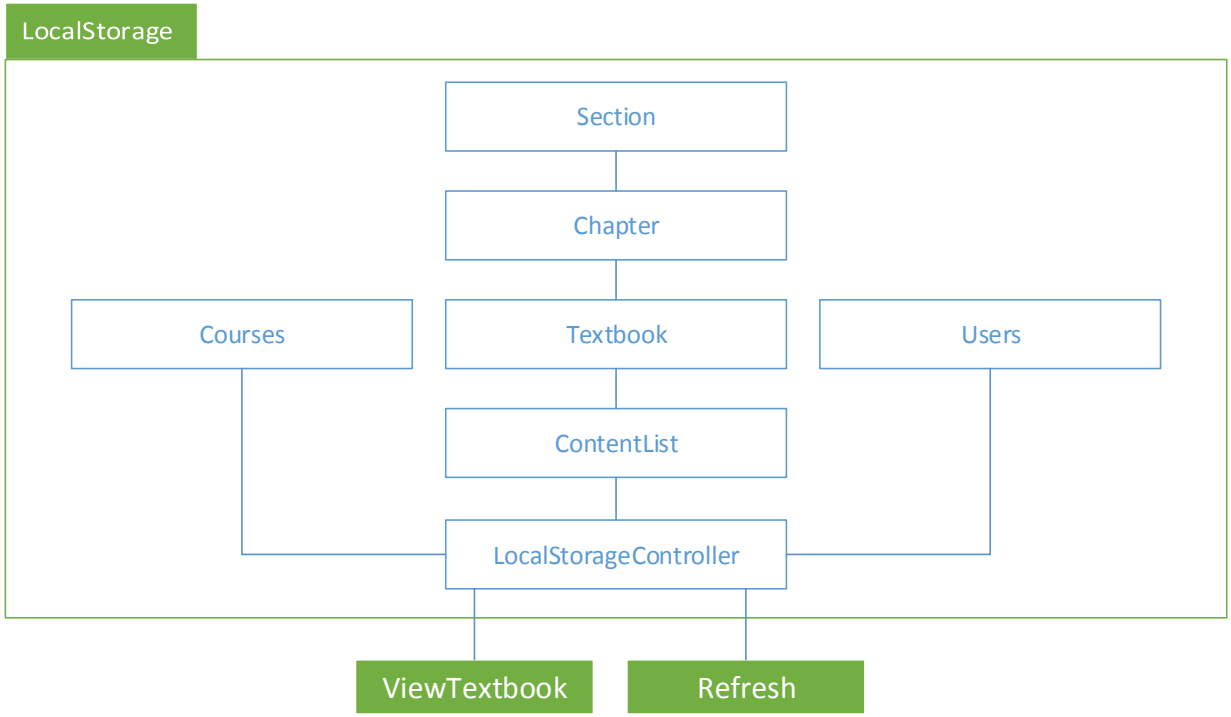

Figure 7 - Local Storage Subsystem

Table 7 - Local Storage Subsystem Traceability

Identifier	[SS-07]
Name	LocalStorage
Traceability	[OB-05], [OB-08], [OB-09], [OB-10], [OB-11], [OB-01]

The Local Storage subsystem is responsible for all handling all storage related functions on the client. Within it, it holds container objects for content (i.e. Textbooks, Chapter, and Sections), the current user and the courses for the aforementioned user. The information within this subsystem is coordinated by the Local Storage Controller and is sent to/updated from the View Textbook and Refresh subsystems respectively.

2.2.6. Refresh

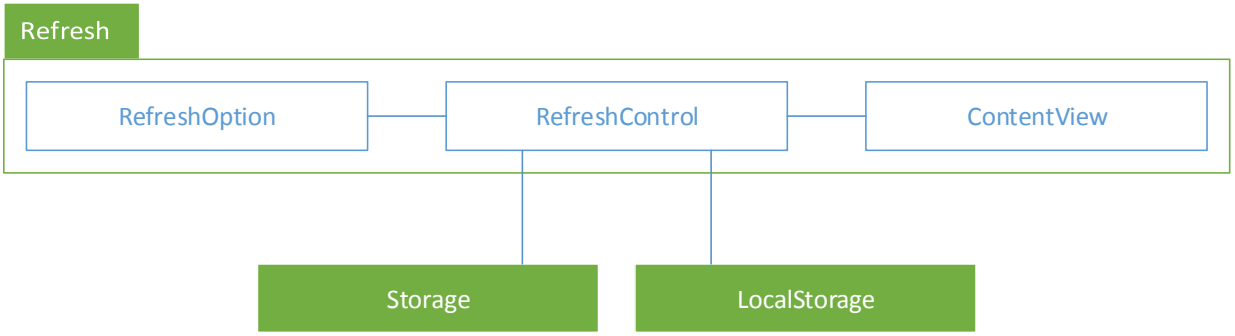

Figure 8 - Refresh Subsystem

Table 8 - Refresh Subsystem Traceability

Identifier	[SS-08]
Name	Refresh
Traceability	[SD-01]

The Refresh subsystem is responsible for updating the Local Storage subsystem with the information retrieved from the Storage subsystem as well as updating all the information that is displayed to the user. This process is initiated by the Refresh Option boundary object but is coordinated through the Refresh Control object.

2.2.7. Textbook Management

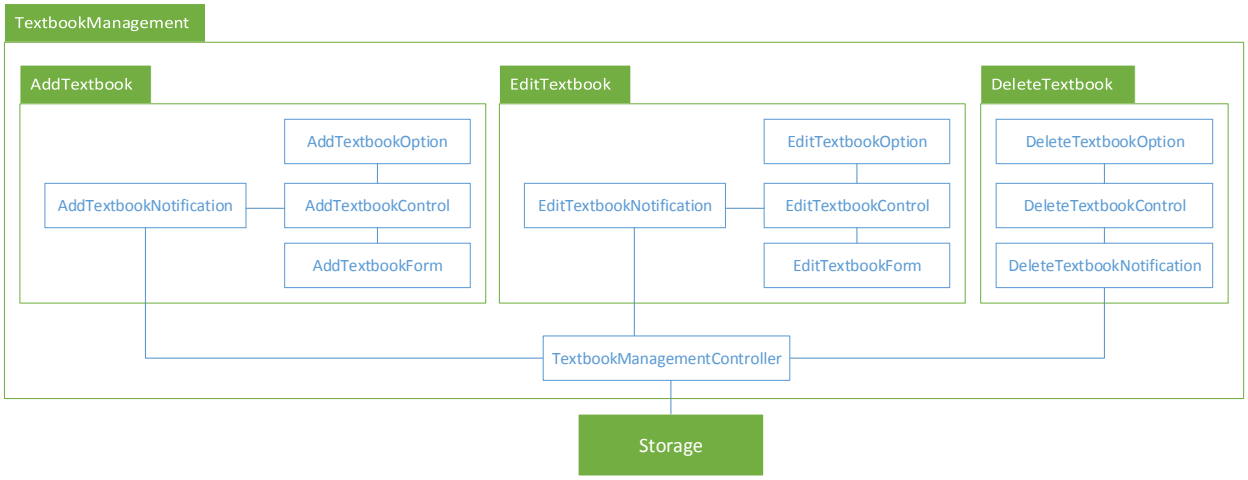

Figure 9 - Textbook Management Subsystem

Table 9 - Textbook Management Subsystem Traceability

Identifier	[SS-09]
Name	TextbookManagement
Traceability	[UC-10], [UC-12], [UC-13]

The Textbook Management subsystem is responsible for adding, updating, and deleting all the details about all the textbooks in the cuTPS system. As such, it is broken down into 3 packages, each with a clear and specific function. Each has a boundary object, control object, and notification object. The Add Textbook and Edit Textbook packages also have Form objects in order to collect all necessary information from the user. The Textbook Management Controller object is responsible for communicating all changes with the Storage subsystem.

2.2.8. Course Management

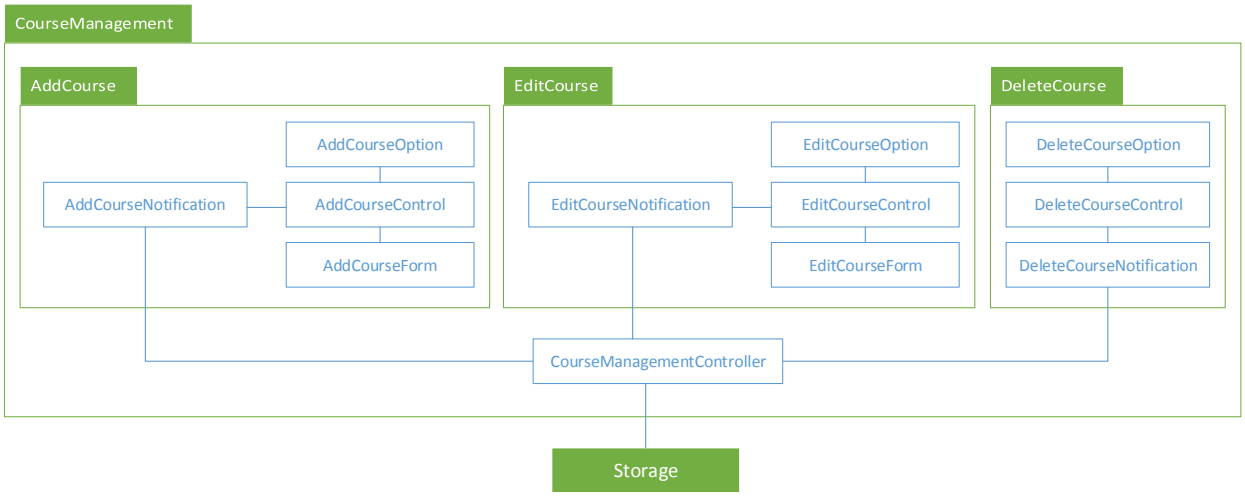

Figure 10 - Course Management Subsystem

Table 10 - Course Management Subsystem Traceability

Identifier	[SS-10]
Name	CourseManagement
Traceability	[UC-15], [UC-17], [UC-18]

The Course Management subsystem is responsible for adding, updating, and deleting all the details about all the courses in the cuTPS system. As such, it is broken down into 3 packages, each with a clear and specific function. Each has a boundary object, control object, and notification object. The Add Course and Edit Course packages also have Form objects in order to collect all necessary information from the user. The Course Management Controller object is responsible for communicating all changes with the Storage subsystem.

2.2.9. User Management

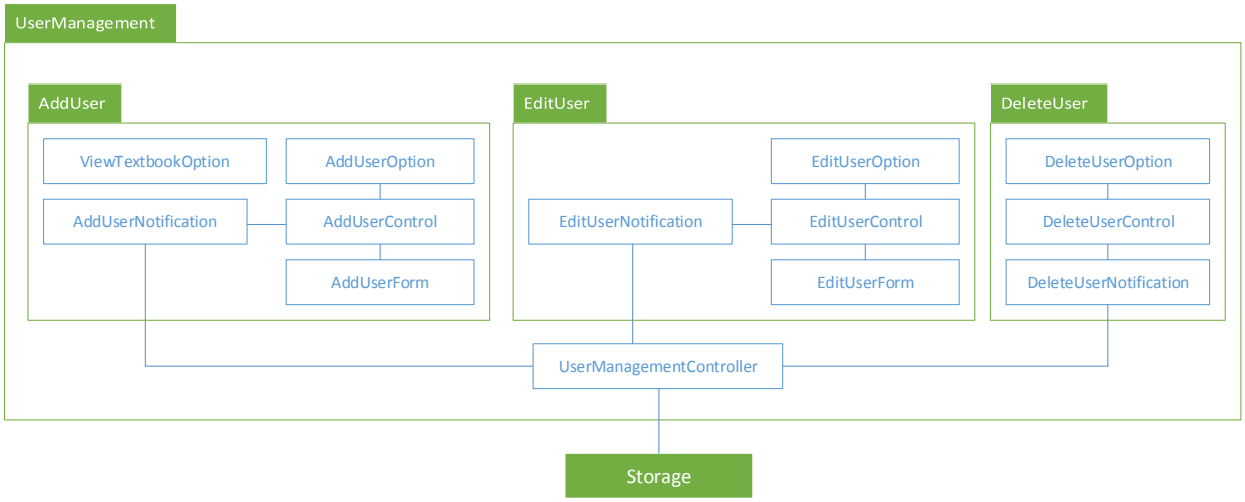

Figure 11 - User Management Subsystem

Table 11 - User Management Subsystem Traceability

Identifier	[SS-11]
Name	UserManagement
Traceability	[F-08]

The User Management subsystem is responsible for adding, updating, and deleting all the details about all the users in the cuTPS system. As such, it is broken down into 3 packages, each with a clear and specific function. Each has a boundary object, control object, and notification object. The Add User and Edit User packages also have Form objects in order to collect all necessary information from the user. The User Management Controller object is responsible for communicating all changes with the Storage subsystem.

2.2.10. Run Report

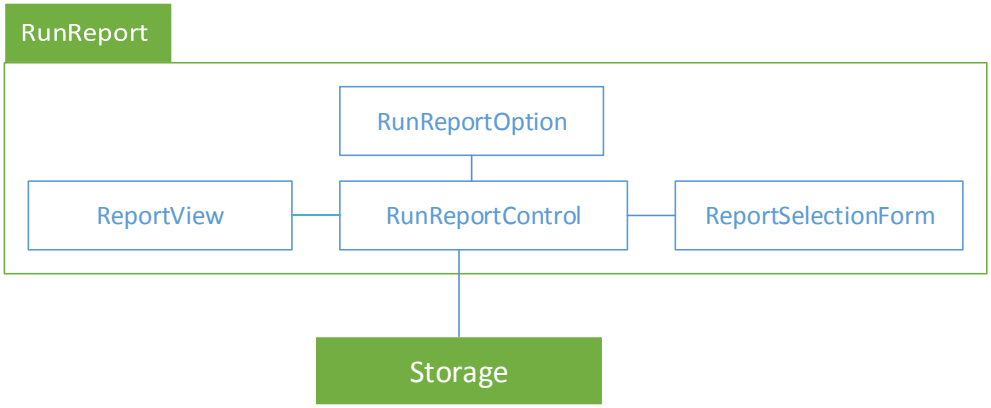

Figure 12 - Run Report Subsystem

Table 12 - Run Report Subsystem Traceability

Identifier	[SS-12]
Name	RunReport
Traceability	[F-10]

The Run Report subsystem is responsible for running and displaying all the different reports available in the cuTPS system. This includes getting the type of report needed from the user, retrieving all information for that report from the Storage subsystem, and displaying all the information to the user. This process is initiated by the Report Selection Form and the Run Report Option boundary objects but is coordinated through the Run Report Control object.

2.2.11. Storage

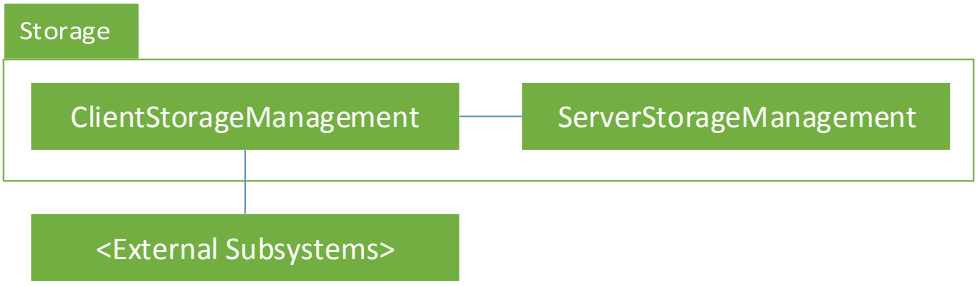

Figure 13 - Storage Subsystem

Table 13 - Storage Subsystem Traceability

Identifier	[SS-12]
Name	Storage
Traceability	[NF-06], [NF-07], [NF-09], [NF-10]

The Storage subsystem is responsible for the retrieval and storage of all cuTPS related data. The Client Storage Management subsystem and Server Storage Management subsystem are each responsible for the serialization and deserialization of message, connecting to each other, and passing the messages between each other. The Server Storage Management subsystem is also responsible for the persistent storage of data (discussed later).

2.2.12. Server Storage Management

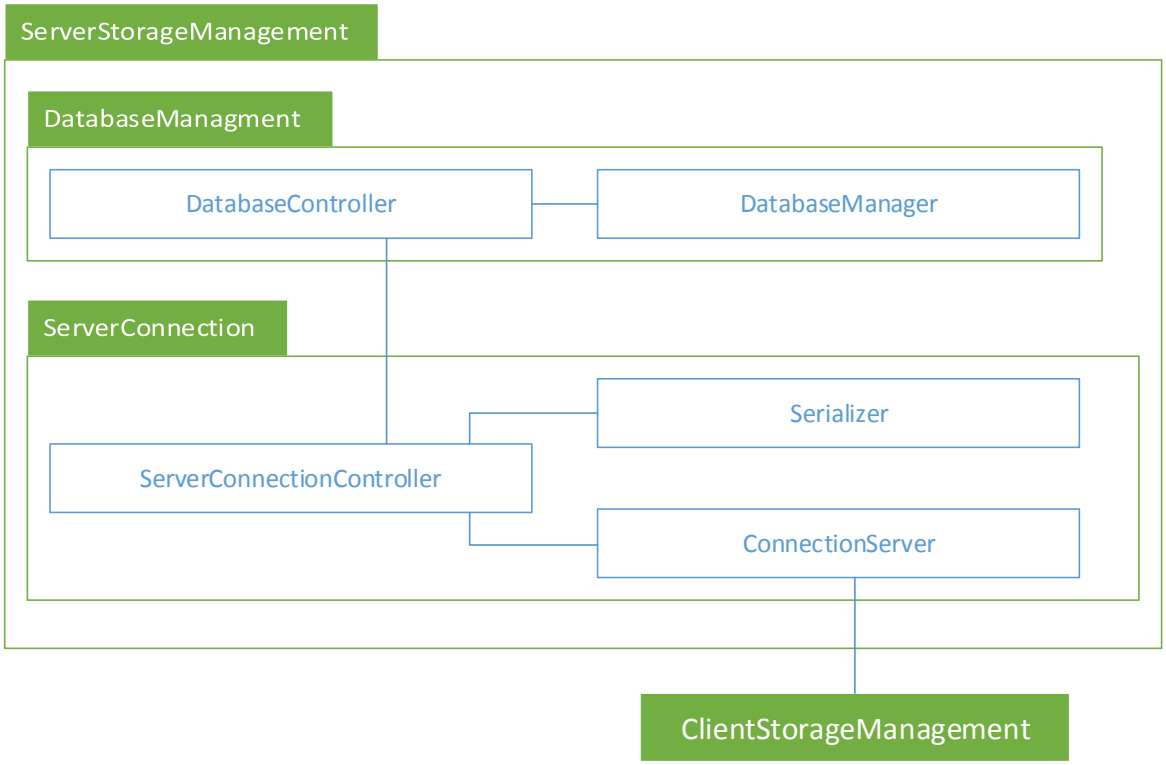

Figure 14 - Server Storage Management Subsystem

Table 14 - Server Storage Management Subsystem Traceability

Identifier	[SS-13]
Name	ServerStorageManagement
Traceability	[NF-06], [NF-07], [NF-09], [NF-10]

The Server Storage Management subsystem is responsible for storage related functions on the server. The Server Connection package handles all message serialization and deserialization as well as managing connections to the Client Storage Management package. The Database Management package is responsible for the retrieval and storage of all persistent data.

2.2.13. Client Storage Management

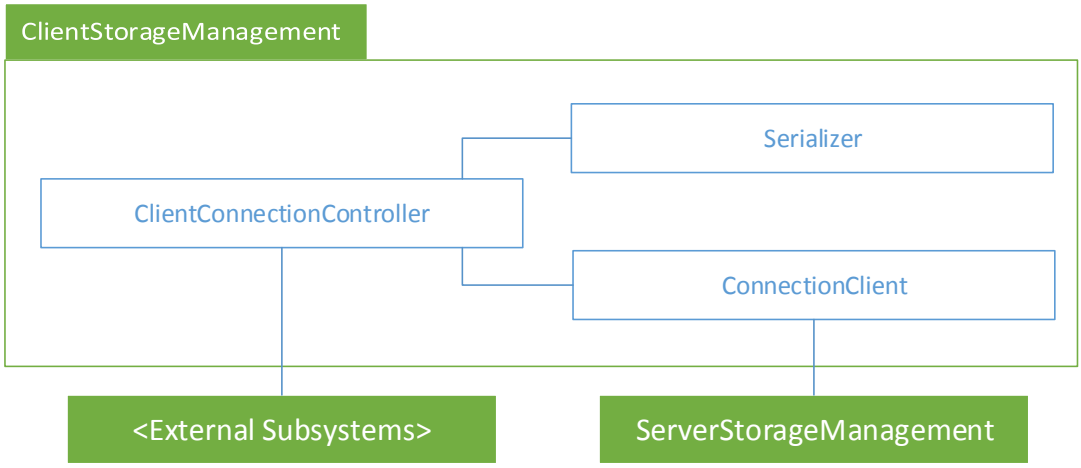

Figure 15 - Client Storage Management Subsystem

Table 15 - Client Storage Management Subsystem Traceability

Identifier	[SS-14]
Name	ClientStorageManagement
Traceability	[NF-06], [NF-07], [NF-09], [NF-10]

The Client Storage Management subsystem is responsible for communicating with the Server Storage Management subsystem and all the external subsystems that link to it. In the interest of keeping Figure 13 from being cluttered they are shown as External Subsystems. There are six in total and they can be seen by looking at section 2.2.1. The Client Storage Management package handles all message serialization and deserialization as well as managing connections to the Server Storage Management package.

2.3. Design Evolution

When we started Deliverable 2, we sat down and designed how we were going to build our system. Our approach to the design was fueled by how we would code it and where the control flow was. Creating classes and then trying to tie them together led to classes being created that were all interdependent. Our system was designed such that if one thing was to be changed, many classes had to be changed.

With our new design we started with a more modular approach. Using our sequence diagrams as a starting point, we drew out all of the objects that we had mentioned in Deliverable 1 and how they were connected based on our sequence diagrams. Once we had them all in front of us we were able to group classes based on functionality. We designed subsystems based on what a group of classes did, and what services they provided.

Compared to our Phase 2 prototype design, our Phase 3 design is far more modular, and simple in its design. In our old one we had one class that controlled the whole system. All actions that were done were done through it. In our new design, systems are separated from one another such that if one were to change, it would leave the others unaffected. Our new system also fits into specific architectural styles and design patterns easily.

3. Design Strategies

3.1. Hardware / Software Mapping

3.1.1. Deployment Diagram Overview

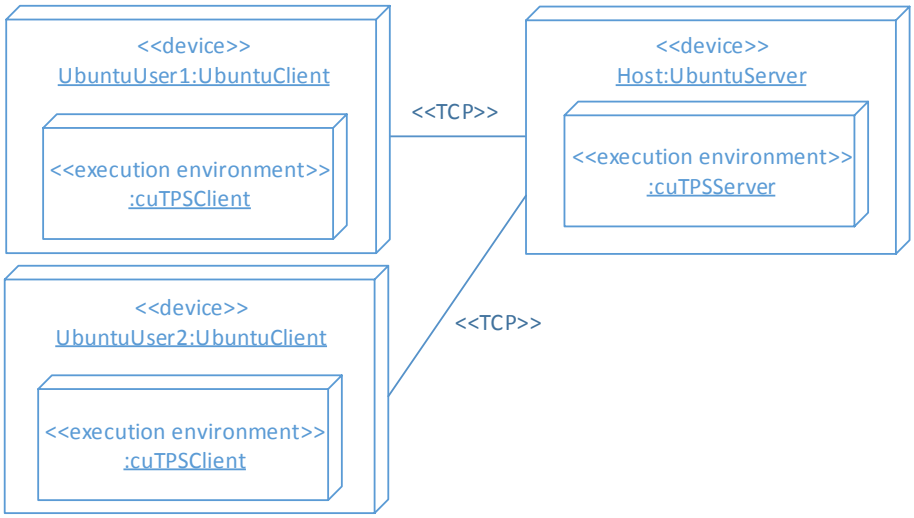

Figure 16 - Deployment Diagram Overview

Table 16 - Deployment Diagram Overview Traceability

Identifier	[DD-01]
Name	NodeOverview
Traceability	[SS-12]

In Figure 16, it is clear that the client-server architecture applies to the physical nodes. The "Host:UbuntuServer" physical node is the server, the ":cuTPSServer" component encompasses all the central database subsystems that provide data and functionality for all clients. Any other nodes would also be a client in the client/server architecture. "UbuntuUser:UbuntuClient" encompasses all the components that are required by the client to retrieve data from the central database.

3.1.2. Deployment Diagram Detailed

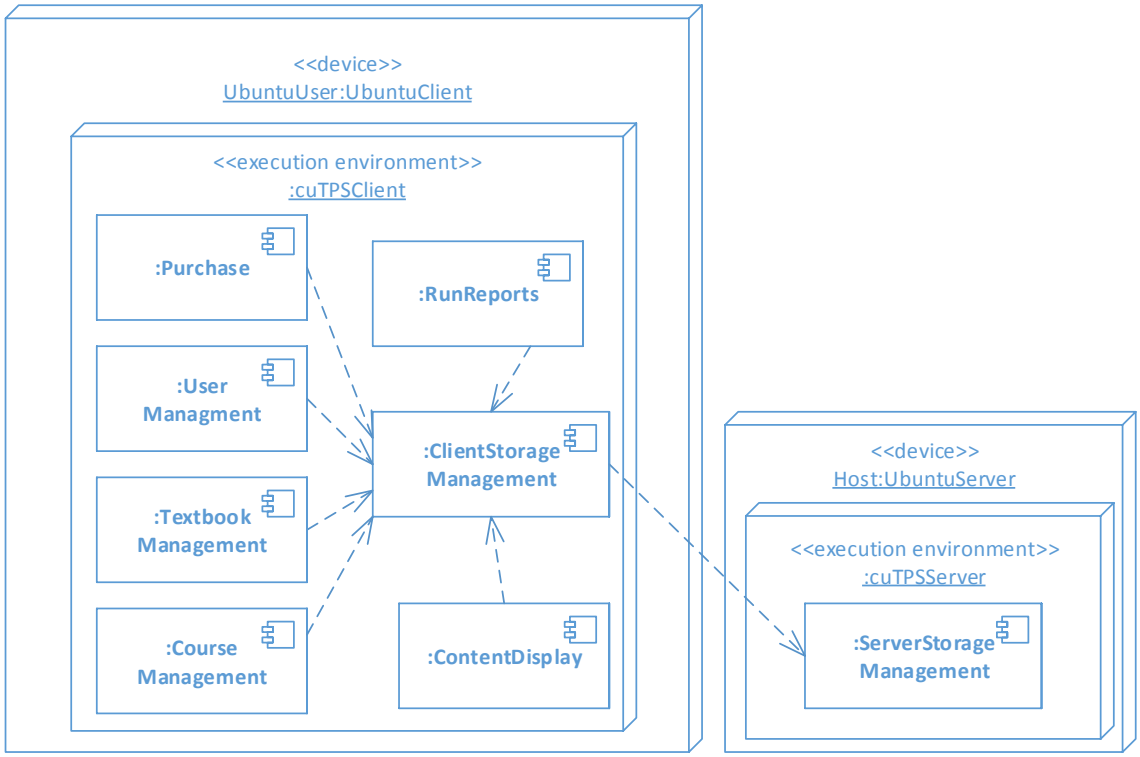

Figure 17 - Deployment Diagram Detailed

Table 17 - Deployment Diagram Detailed Traceability

Identifier	[DD-02]
Name	ComponentNodeOverview
Traceability	[SS-03]

The repository architecture style is also used. This is shown in Figure 17 as all the client components on the client physical node “UbuntuUser:UbuntuClient” interact with the “ClientStorageManagement” component. The repository architecture is also shown in Figure 3. All top layer subsystems are associated and connected with one subsystem, which is the “storage” subsystem. Storage encapsulates both “ClientStorageManagement” and “ServerStorageManagment”, providing an interface that abstracts storage and the fact that it is on a server.

3.2. Persistent Data Management

3.2.1. Overview

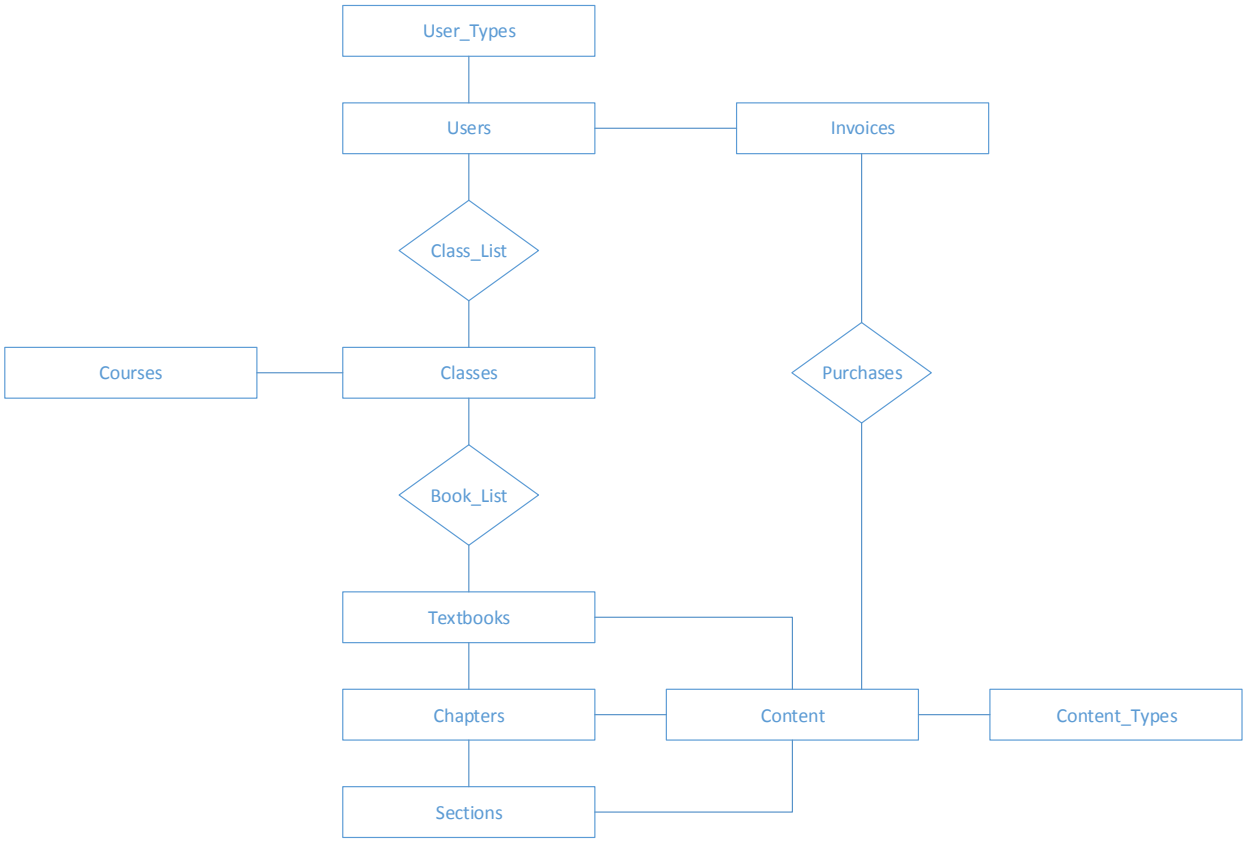

Figure 18 - Persistent Data Management Overview

Table 18 - Persistent Storage Traceability

Identifier	[DB-01]
Name	Persistent Data Management Overview

Figure 18 is an ER diagram depicting the persistent data management strategy of the cuTPS system. It is not an ER diagram in the traditional sense as it only shows the relationships between the tables and not the attributes of each table. The attributes and would unnecessarily clutter the diagram and they are described in detail below.

3.2.2. User Types

Table 19 - User Types

Column	Type	Attributes
type	Text	Primary key

The User Types table stores the types of users that exist in the system. As per the cuTPS requirements from Deliverable 1 the only types that would exist in this table are student, content manager and administrator however in the future if any other type of user needs to be added or an existing type renamed then they would simply add to or modify this table.

Table 20 - User Types Table Traceability

Identifier	[DD-02]
Name	User Types
Traceability	[OB-01], [OB-02], [OB-03], [OB-04]

3.2.3. Users

Table 21 - Users

Column	Type	Attributes
username	Text	Primary key
password	Text	Not null
type	Text	Foreign key
name	Text	Not null

The Users table stores all the User instances that exist in the system. Each User is identified by its username so it is the primary key of the table. A User also has a password, used for authentication, and a name, to store their full name. The type field determines what type of user they are and it is a foreign key to the User Types table. There was no requirement to store any other information about a User however in the future if, for example, there was a requirement to store a student's student number,

then a Student table would be created to hold all student-specific attributes with a foreign key dependency on the User's table username field.

Table 22 - Users Table Traceability

Identifier	[DD-03]
Name	Users
Traceability	[OB-01]

3.2.4. Courses

Table 23 - Courses

Column	Type	Attributes
code	Text	Primary key
name	Text	Not null

The Courses tables stores all the Course instances that exist in the system. Each course is identified by its code (e.g. COMP3004) so it is the primary key of the table. Each course also has a course name to describe the course (e.g. Introduction to Software Engineering).

Table 24 - Courses Table Traceability

Identifier	[DD-04]
Name	Users
Traceability	[OB-05]

3.2.5. Classes

Table 25 - Classes

Column	Type	Attributes
id	Integer	Not null, unique
semester	Text	Primary key
course	Text	Primary key, foreign key

The Classes table stores all the Class instances that exist in the system. A class is made up of a semester (e.g. Fall 2014) and a Course (e.g. COMP3004), and together they make up the primary key of the table. This way a course can appear many times in this table but only once per semester. Each instance of a class is given a unique (to this table) number which serves to enforce relationships in other tables. The course column in this table is foreign key dependency on the code column of the Courses table.

Table 26 - Classes Table Traceability

Identifier	[DD-05]
Name	Users
Traceability	[OB-05]

3.2.6. Class List

Table 27 - Class List

Column	Type	Attributes
student	Text	Primary key, foreign key
class	Text	Primary key, foreign key

The Class List table stores which students are in which class. Both fields are foreign key dependencies on other tables and together they make up the primary key of this table. The student column references the username column of the Users table and the class column references the id column of the Classes table.

Table 28 - Class List Table Traceability

Identifier	[DD-06]
Name	Class List
Traceability	[OB-04], [OB-05]

3.2.7. Content Types

Table 29 - Content Types

Column	Type	Attributes
type	Text	Primary key

The Content Types table stores the type of content that is available in the system. . As per the cuTPS requirements from Deliverable 1 the only types that would exist in this table are textbook, chapter, and section. In the future if any other type of content is needed then it would simply need to be added to this table.

Table 30 - Content Types Traceability

Identifier	[DD-07]
Name	Content Types
Traceability	[OB-08], [OB-09], [OB-10], [OB-11]

3.2.8. Content

Table 31 - Content

Column	Type	Attributes
id	Integer	Primary key
type	Text	Foreign key

The Content table provides unique identifiers to each piece of content that is available in the system. Since there are multiple kinds of content, each in their own separate tables, this table allows for a simple and easy way to uniquely identify each piece of content since the database can guarantee that that the id will be unique and it can provide new ids very quickly. The type column is a foreign key

dependency on the type column of the Content Types table. Each id is associated with a content type so that, given only a content id, the system knows which table to query to get the information about that piece of content without searching through all of them. While this means getting the information about a piece content, given only the content id, will involve performing two separate queries (the first to get the type of content, the second to query that type of content's table), in the long term as the amount of information in the database grows, performing two queries will be faster than searching all the tables.

Table 32 - Content Table Traceability

Identifier	[DD-08]
Name	Content
Traceability	[OB-08]

3.2.9. Textbooks

Table 33 - Textbooks

Column	Type	Attributes
isbn	Text	Primary key
title	Text	Not null
publisher	Text	Not null
author	Text	Not null
year	Integer	Not null
edition	Text	
description	Text	
availability	Integer	Not null
price	Real	Not null
content_id	Integer	Foreign key

The Textbooks table stores all the Textbook instances that exist in the system. A textbook is identified by its isbn number so it is the primary key of this table. The availability column of this table represents whether the Textbook is available for sale or not. As SQLite does not have a Boolean data type an integer is used in its stead and will be either a 1 or 0. The content_id is a foreign key dependency on the id column of the Content table.

Table 34 - Textbooks Table Traceability

Identifier	[DD-09]
Name	Textbook
Traceability	[OB-08], [OB-09]

3.2.10. Chapters

Table 35 - Chapters

Column	Type	Attributes
name	Text	Not null
number	Integer	Primary key
textbook	Text	Primary key, foreign key
description	Text	
availability	Integer	Not null
price	Real	Not null
content_id	Integer	Foreign key

The Chapters table stores all the Chapter instances that exist in the system. A chapter is identified by its number and the textbook it belongs to. The combination of these two columns make the primary key of this table. The availability column of this table represents whether the Chapter is available for sale or not. As SQLite does not have a Boolean data type an integer is used in its stead and will be either a 1 or 0. The content_id is a foreign key dependency on the id column of the Content table and the Textbook is a foreign key dependency on the isbn column of the Textbooks table.

Table 36 - Chapters Table Traceability

Identifier	[DD-10]
Name	Chapter
Traceability	[OB-08], [OB-10]

3.2.11. Sections

Table 37 - Sections

Column	Type	Attributes
name	Text	Not null
number	Integer	Primary key
chapter	Integer	Primary key, foreign key
textbook	Text	Primary key, foreign key
description	Text	
availability	Integer	Not null
price	Real	Not null
content_id	Integer	Foreign key

The Sections table stores all the Section instances that exist in the system. A Section is identified by its number, the Chapter it belongs to, and the Textbook it belongs to. The combination of these three columns make the primary key of this table. The availability column of this table represents whether the Chapter is available for sale or not. As SQLite does not have a Boolean data type an integer is used in its stead and will be either a 1 or 0. The content_id is a foreign key dependency on the id column of the Content table, the chapter column is a foreign key dependency on the number column of the Chapters table, and the textbook column is a foreign key dependency on the isbn column of the Textbooks table.

Table 38 - Sections Table Traceability

Identifier	[DD-11]
Name	Section
Traceability	[OB-08], [OB-11]

3.2.12. Book List

Table 39 – Book List

Column	Type	Attributes
textbook	Text	Primary key, foreign key
class	Integer	Primary key, foreign key

The Book List table stores which textbooks are for which class. Both fields are foreign key dependencies on other tables and together they make up the primary key of this table. The textbook

column references the isbn column of the Textbooks table and the class column references the id column of the Classes table. As class has a semester as one of its attributes, this table allows for different textbooks to be assigned to the same course for different semesters or different courses in the same semester. Also, as this design only links textbooks to classes, all of the chapters and sections that belong to that textbook are also linked to that class.

Table 40 - Book List Table Traceability

Identifier	[DD-12]
Name	Book List
Traceability	[OB-05], [OB-08]

3.2.13. Invoices

Table 41 – Invoices

Column	Type	Attributes
id	Integer	Primary key
student	Text	Foreign key
date_purchased	Text	Not null

The Invoices table stores all the Invoice instances that exist in the system. The id is a unique integer that is generated by the database that serves as the primary key of this table. The student column is a foreign key dependency on the username column of the Users table that represents which User this invoice belongs to. The date_purchased column is the date the purchase was made.

Table 42 - Invoices Table Traceability

Identifier	[DD-13]
Name	Invoices
Traceability	[OB-12]

3.2.14. Purchases

Table 43 – Purchases

Column	Type	Attributes
invoice_id	Integer	Primary key, foreign key
content_id	Integer	Primary key, foreign key
purchase_price	Real	Not null

The Purchases table stores which content items were purchased for which Invoice. The invoice_id and content_id fields are both foreign key dependencies on the id column of the Invoices table and the id column of the Content table respectively, and together they make the primary key of this table. The purchase_price column stores the price at which the content item was bought at. This column was added in order to allow a content manager to update the price of a piece of content without affecting any invoices in the process. This design allows for different multiple content items to belong to the same invoice and for the same content item to belong to multiple invoices at different prices.

Table 44 - Purchases Table Traceability

Identifier	[DD-14]
Name	Purchases
Traceability	[OB-08], [OB-11]

3.3. Design Patterns

The new subsystem decomposition uses two design patterns: the façade design pattern and the composite design pattern. The façade design pattern is used multiple times. When any subsystem interacts with the storage subsystem it uses a façade class (Storage Controller) to abstract the setting and retrieval of information from the persistent storage. It is also used when the checkout subsystem retrieves information from the Shopping Cart Management subsystem, and when the Refresh and View Textbook subsystems interact with the Local Storage subsystem.

The composite design pattern will be used in the client's GUI functionality. Classes like QButton that inherit QWidget will be used inside of other QWidget classes like QDialog. This allows an update like a resize to take place in all of the QWidgets at once. It also allows for the creation of new QWidgets that can be used in the same place as other ones but they can behave very differently.

4. Subsystem Services

4.1.1. Subsystem Services Overview

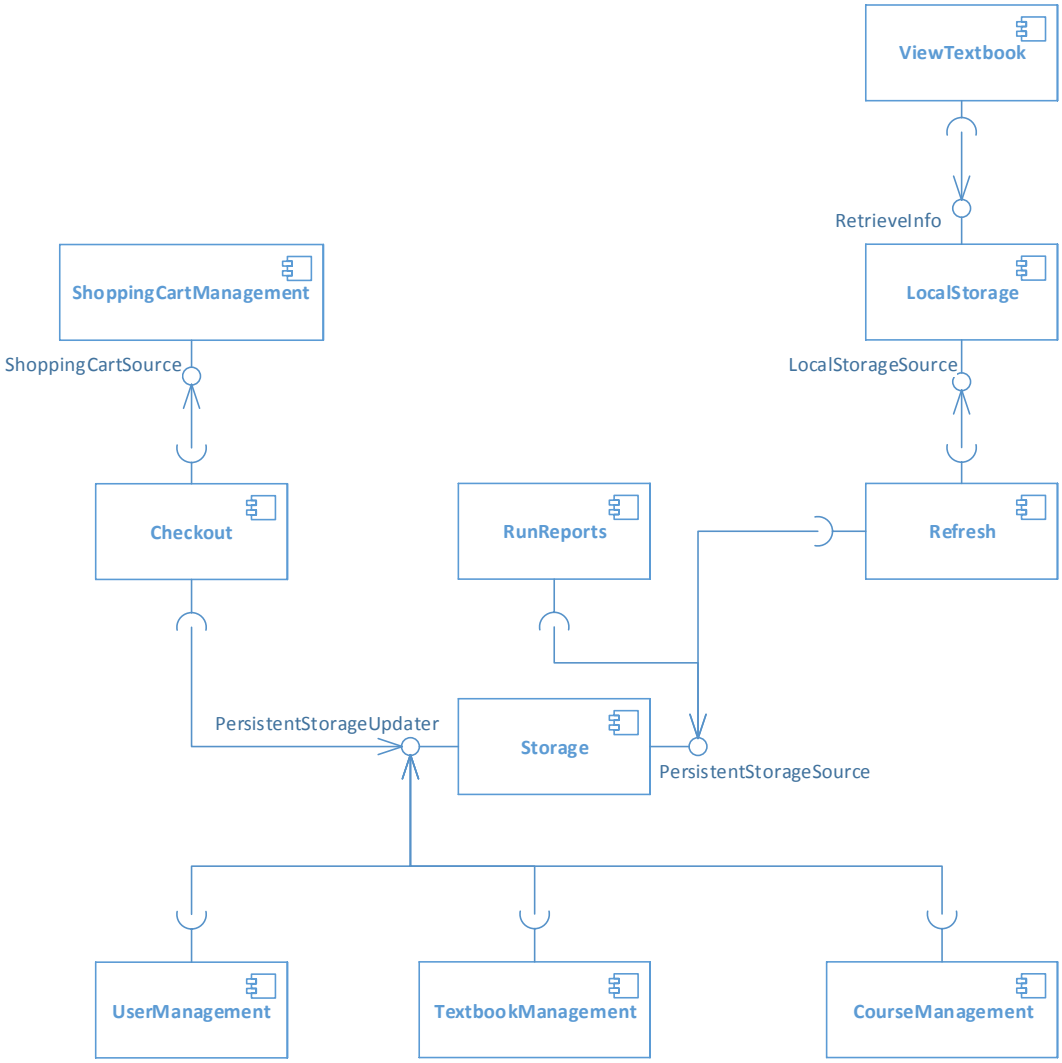

Figure 19 - Subsystem Services Overview

Table 45 - Subsystem Services Overview Traceability

Identifier	[SE-01]
Name	SubSystem Services Overview
Traceability	[SS-03]

Table 46 - Subsystem Services Overview

Subsystem	**Service**
ViewTextbook [SE-02]	Does not provide any services
LocalStorage [SE-03]	LocalStorageUpdater: - Provides the ability to update LocalStorage LocalStorageSource: - Functions to retrieve information stored on the client LocalStorageControl provides two operations, one for each services [SS-07]
Refresh [SE-04]	Does not provide any services
Storage [SE-05]	PersistentStorageUpdater: - Provides functions to store information in persistent storage PersistentStorageSource: - Provides functions to retrieve information from persistent storage ClientConnectionControl provides operations for each of the different types of update (create textbook, delete user, etc…) and each type of retrieval (get report, get content list, etc) [SS-12], [SS-13]
RunReports [SE-06]	Does not provide any services
ShoppingCartManagement [SE-07]	ShoppingCartSource: - Provides the content currently in the shopping cart ShoppingCartControl provides an operation for retrieving the contents of the shopping cart. [SS-04]
Checkout [SE-08]	Does not provide any services
UserManagement [SE-09]	Does not provide any services
TextbookManagement [SE-10]	Does not provide any services
CourseManagement [SE-11]	Does not provide any services

4.1.2. Storage Services

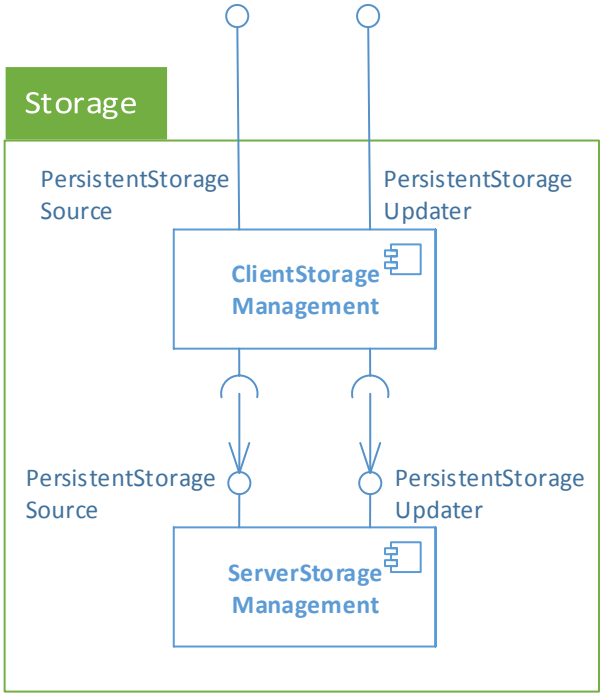

Figure 20 - Storage Services

Table 47 - Storage Services

Subsystem	**Services**
ClientStorageManagement [SE-05]	PersistentStorageUpdater: - Provides functions to store information in persistent storage PersistentStorageSource: - Provides functions to retrieve information from persistent storage ClientConnectionControl provides operations for each of the different types of update (create textbook, delete user, etc…) and each type of retrieval (get report, get content list, etc) [SS-12], [SS-13]

ServerStorageManagement [SE-12]	PersistentStorageUpdater: - Provides operations to receive data from the client and store it in the database PersistentStorageSource: - Provides functions to retrieve information from persistent database and pass it to the client ConnectionServer provides an interface for all incoming and outgoing messages for the server. [SS-12], [SS-14]

5. Class Interfaces

5.1.1. Shopping Cart Interface

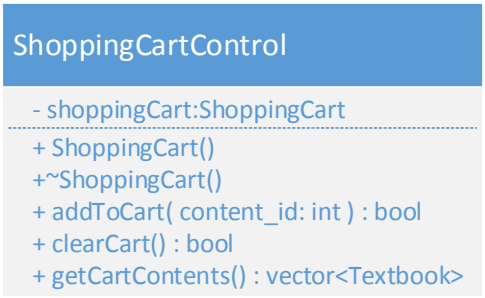

Figure 21 - Shopping Cart Interface

The ShoppingCartManagment subsystem provides one service, the ability to retrieve the contents of the shopping cart. This services is provided by the function, getCartContents(), which returns a vector of textbooks.

Table 48 - Shopping Cart Control Traceability

Identifier	[CI-01]
Name	ShoppingCartControl
Traceability	[SE-07]

5.1.2. Local Storage Management

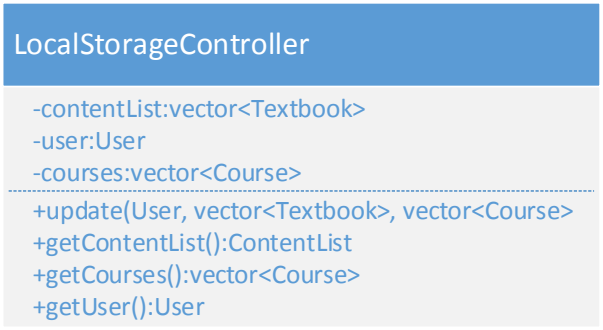

Figure 22 - Local Storage Controller Interface

The LocalStorageManagment subsystem provides two services. The service LocalStorageUpdater is provided by the function update(User, vector<Textbook>, vector<Course>). This function updates the localstorage with the current content associated with the user and the courses that the user is enrolled in. If the user is a content manager they are considered to be enrolled in all of the courses and can see all of them.

LocalStorageManagment also provides the LocalStorageSource service. This service is provided by the functions get ContentList(), getCourses(), and getUser(). These functions allow other subsystems to retrieve the information currently kept track of on the client.

Table 49 - Local Storage Controller Traceability

Identifier	[CI-02]
Name	LocalStorageController
Traceability	[SE-03]

5.1.3. Client Connection Controller

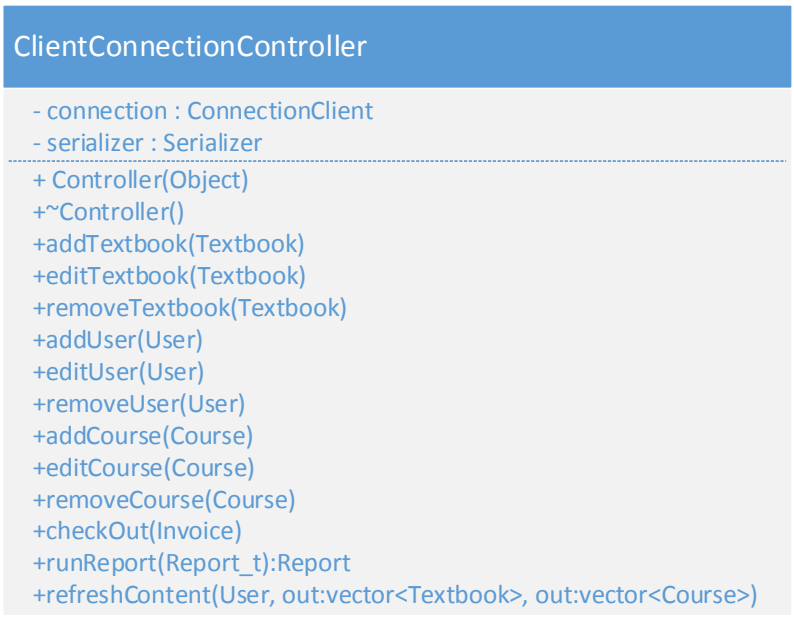

Figure 23 - Client Connection Controller Interface

The Storage subsystem also has two services. The PersistentStorageUpdater service provides the functions, addTextbook(), addUser(), addCourse(), editTextbook(), editUser(), editCourse(), removeTextbook(), removeUser(), removeCourse(), and checkout(). These functions provided by the class ClientConnectionController give the ability to change the contents of the persistent storage. The service, PersistentStorageSource, provides the functions, refreshContent(), and runReport. Refresh content takes the currently logged in User and returns the list of courses they are enrolled in and the textbooks associated with those courses. The report takes the wanted report type and returns the request report.

Table 50 - Client Connection Controller Traceability

Identifier	[CI-03]
Name	ClientConnectionController
Traceability	[SE-05]

5.1.4. Server Storage Management

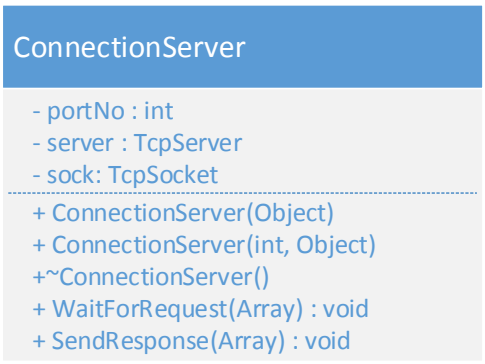

Figure 24 - Server Storage Management Interface

The ServerStorageManagement subsystem provides the same services as the ClientStorageManagement subsystem. All of the operations provided that are part of the services of ClientStorageManagement are passed to the ConnectionServer. The function WaitForRequest() receives the intent of the client and sendResponse() sends the response.

Table 51 - Server Storage Management Traceability

Identifier	[CI-04]
Name	ConnectionServer
Traceability	[SE-12]