CS3354.004 Software Engineering
Final Project Deliverable 2

Self-Use Shipping Kiosk Software

Aashna Pathi, Ann Jayan, Cristina Adame, Namra Zubair, Nivedha Sreenivasan, Sanjana Kotha, Shashi Rajesh, Susan Zhang

Section 0: GitHub Repository

Github Repository Link : https://github.com/namra95/CS3354-Shipping-Kiosk.git

- Deliverable #2 Report: Sanjana Kotha
- Unit Test Code: Nivedha Sreenivasan
- Presentation Slides: Susan Zhang

Section 1: Delegation of Tasks

Aashna Pathi will install Git Bash on my environment and work on defining the non-functional requirements for the project for *Deliverable 1*. For *Deliverable 2*, Aashna will work on the Project Scheduling aspect of the proposed software, contribute to the conclusion, and contribute to the Group Presentation Slides.
Ann Jayan will install Git Bash on my environment, create a GitHub account, address feedback from the final project proposal, and create a Class Diagram for *Deliverable 1*. For *Deliverable 2*, Ann will work revising the sequence diagram from Deliverable, she will push the final project Deliverable #2 Report to GitHub, contribute to the conclusion, and contribute to the Group Presentation Slides.
Cristina Adame will work on the functional requirements for *Deliverable 1*. For *Deliverable 2*, Cristina will work on Estimated Cost of Hardware Products, collaborate with Ann for the estimation study, contribute to the conclusion, and contribute to the Group Presentation Slides.
Namra Zubair will make a GitHub repository for this project and add my teammates as collaborators (Task 1.2 & Task 1.3) and work on preliminary research about the proposed topic for *Deliverable 1*. For *Deliverable 2*, Namra will work on Cost, Effort, and Pricing Estimation, the Estimated Cost of Software Products, contribute to the conclusion, and contribute to the Group Presentation Slides.
Nivedha Sreenivasan will install Git Bash on my environment, create a GitHub account, research government-based rules, and create a sequence diagram for *Deliverable 1*. For *Deliverable 2*, Nivedha will work on the test code, create a test plan for software, push the Unit test code for a sample unit to GitHub, contribute to the conclusion, and submit the group proposal and group-related documentation via eLearning.
Sanjana Kotha will install Git Bash on my environment, create a GitHub account and create the case diagram for *Deliverable 1*. For *Deliverable 2*, Sanjana will perform comparisons with similar designs and contribute to the conclusion.
Shashi Rajesh will install Git Bash on my environment and make a second commit of the project scope (Task 1.5) for *Deliverable 1*. For *Deliverable 2*, Shashi will work on Estimated Cost of Personnel, collaborate with Ann for the estimation study, contribute to the conclusion, and contribute to the Group Presentation Slides.
Susan Zhang will install Git Bash on my environment, create a GitHub account, create the README.md file (Task 1.4), record the delegation of tasks, address feedback from Final Project Proposal, and create the Architectural Design diagram for *Deliverable 1*. For *Deliverable 2*,

Susan will record the delegated task list, record Deliverable #1's content, create the Group Presentation Slides, create the demo of user interface design that shows screen to screen, and push the Presentation slides to GitHub repository.

Section 2: Project Deliverable 1 Content

Section 2.1: Project Overview

Project Goals

Our project goals revolve around creating a software that measures and takes package details as input, determines/dispenses necessary wrapping items, processes shipping details, delivers a shipping label, and processes payment. The goals are to streamline the shipping process, create a safer shipping process in a post-COVID-19 world, and expand the software by deploying kiosks in popular public establishments: libraries, grocery stores, malls, and mail collection boxes.

Project Motivations

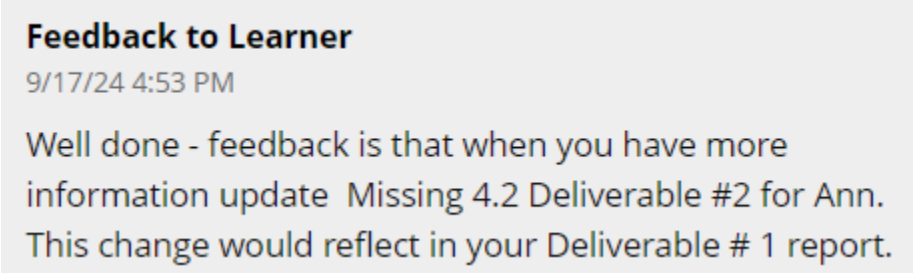
Our motivation to choose a Self-Use Shipping Kiosk software as our project stemmed from our own personal frustrations with the postal office. The current self-service kiosks are outdated, complex, and often out of service. In-person postal office booths are burdened with long lines, which are considerably inconvenient, time consuming and especially uncomfortable when attempting to ship multiple packages. Our group aims to streamline the shipping & mailing process through our software in a sequentially logical and prompt manner. Additionally, we were inspired to create a self-service software due to its popularity – in terms of both efficiency and safety – post-COVID-19.
In the real world, we expect our kiosks to be located in post offices. A stretch goal location could be inside establishments (libraries, grocery stores, malls, etc.) near mail collection boxes.

Task Delegation

See Final Project Deliverable 2: Section 1: Delegation of Tasks

Proposal Feedback

Our given proposal feedback is:



After meeting with Ann, her Deliverable #2 tasks will include contributing to the Group Presentation Slides, adding/pushing/committing any changes to our GitHub repository, contributing to our report conclusion, and contributing to our report references, in addition to working on Project Scheduling, Cost Effort, Pricing Estimation, Duration and Staffing.

Government-Based Rules

The system has to follow data privacy laws like GDPR for EU users and CCPA for U.S [1]. users. In other words, it should only collect essential information such as one's name, address, and payment details. Moreover, before gathering or processing any personal data, the system needs to get the user's explicit consent, so they know exactly what's happening with their information. Also, the system should not hold onto the user's data for longer than it needs to. Once the data no longer needs to be stored, it should be securely deleted. When data is being transmitted, it should be encrypted (using SSL/TLS) to keep it safe from anyone trying to intercept it. Lastly, the user should have the right to see the data that's been collected about them and be able to request its deletion if they want it gone [2].
The system should follow environmental guidelines to help reduce packaging waste. This means encouraging users to pick minimal packaging options or providing reusable materials whenever possible. It's also essential that any materials the kiosk provides, like labels and boxes, are recyclable so they don't just end up in the trash. Following local and federal recycling regulations is key here, as it helps make sure the system supports sustainability and reduces its environmental impact.

Section 2.2: GitHub Repository

Task 1.3: Namra Zubair
Task 1.4: Susan Zhang
Task 1.5: Shashi Rajesh

Github Repository Link : https://github.com/namra95/CS3354-Shipping-Kiosk.git

Section 2.3: Delegation of Tasks

See Final Project Deliverable 2: Section 1: Delegation of Tasks

Section 2.4: Software Process Model

Since the product will be deployed directly to the public, it will require high levels of use, security and frequent updates to respond to dynamic user needs and technological changes. As such, an incremental process model along with core agile principles will be most suitable for this project.
Initially, a limited number of kiosks will be rolled-out to the public with basic features of shipping services being offered. During this first phase, user feedback will be heavily-relied upon to enhance the UI and rapidly introduce newer features and/or to eliminate older features.

Working in increments would allow the initial phases to function smoothly, and the use of agile methodologies would mean developers can easily refine the product based on user feedback to ensure the kiosks are functional and easy to use, and then rapidly deploy those changes in a frequent manner.
In comparison, a waterfall or v model approach is not suitable as it would introduce rigidity, a need for heavy documentation, and constant testing in the development process. Moreover, it will be extremely costly to develop and would require a lot of initial funding, which is not a feasible option for the development team.

Below is a summary of what the initial development sprint phase could potentially look like:

1. Requirements Gathering
    - Describing main features being offered, such as label printing, shipment tracking, collecting payments, a smooth and interactive UI/UX
    - Security considerations with regard to payment processing, protecting sensitive user information (user name, addresses, etc.)
    - Collaboration with third-party shipping carriers (e.g., FedEx, UPS)
2. Design and Development
    - System architecture design
    - UX/UI design considerations
    - Incremental feature releases
    - Defining APIs for payment and shipment handling
3. Testing
    - Ensuring payment system and shipment tracking function smoothly
    - Testing product with real users to ensure functionality and product effectiveness
4. Deployment and Maintenance
    - Rapid and frequent release of software updates
    - Ensuring contingencies are in place in the event of system failure
    - Gathering user feedback for the next development phase
    - Maintaining provisions for post-deployment support to users

Section 2.5: Software Functional Requirements

1. User shall be able to select package type and input package dimensions
    - ❖ Package Types: Legal Document/Flat, Small, Medium, Large, Extra Large
    - ❖ Dimensions: Length, Width, Weight
2. The system shall be able to automatically determine the package materials, from the package specifications, and dispense the materials
    - ❖ Materials: Cardboard Mailer, Cardboard Box, Bubble Wrap, Tape, Fragile Label
3. The system shall be able to validate an entered address for shipping
4. The system shall be able to generate and print a valid shipping label with all necessary information

- ❖ Information: Mailer Information, Recipient Information, Shipping Address, Tracking Barcode

5. User shall be able to view notary availability and book an appointment
6. The system shall be able to process multiple packages in a single continuous transaction
7. The system shall be able to calculate final cost and process payment

Section 2.6: Software Non-Functional Requirements

1. Product Requirements

1.1. Usability Requirements

- ❖ The Shipping Kiosk system should be available to all users from Monday to Friday from 8:00 AM to 8:00 PM.
- ❖ The Shipping Kiosks will be located in all post offices in the United States (will be expanded globally in the future).
- ❖ All kiosks will be self-service and have a touchscreen display.
- ❖ All users will be presented with a 15 second video on how to use the kiosk before beginning any transaction.

1.2. Efficiency Requirements

1.2.1. Performance Requirements

- ❖ The Shipping Kiosk system should perform all processing steps (validate address and payment) within a reasonable amount of time (3-5 seconds).
- ❖ The system must print the shipping label and dispense packaging materials immediately for the user.
- ❖ The system must be able to handle a large number of transactions with little latency to promote scalability.

1.2.2. Space Requirements

- ❖ The database must be able to store name, address, and payment details for thousands of users (can be adjusted based on volume of customers using the software at any given time).

1.3. Dependability Requirements

- ❖ The system must include provisions for real-time monitoring of software or hardware issues.
- ❖ If the system fails, an error message must immediately be sent to the developer team..

1.4. Security Requirements

- ❖ The Shipping Kiosk system should only collect basic information from package senders (name, address, payment details).
- ❖ The system must securely delete that is no longer needed.
- ❖ The system must encrypt data (using SSL/TLS) during data transmission.

2. Organizational Requirements

2.1. Environmental Requirements

- ❖ The Shipping Kiosk system will encourage users to make use of reusable, recyclable packaging materials that are environmentally friendly.
- ❖ The system will calculate the smallest possible package dimensions to reduce unnecessary waste.
- ❖ All labels and packaging materials provided by the Shipping Kiosk will provide information about disposal to ensure the paper gets reused and recycled.
- ❖ The paper and packaging materials will be made of recycled materials.

2.2. Operational Requirements

- ❖ The Shipping Kiosk software must be checked for software or hardware issues on a bi-weekly basis.
- ❖ Any operational issues must be immediately addressed.
- ❖ Adequate paper for printing shipping labels must be present and replaced as needed by the post office it is placed in.
- ❖ The system should include customer support during all hours of operation.

2.3. Development Requirements

- ❖ There should be a system to provide real-time alerts for any software or hardware malfunctions.

3. External Requirements

3.1. Regulatory Requirements

- ❖ The Shipping Kiosk software must follow data privacy laws for the country it is being used in (currently only in the United States).
- ❖ In the United States, the software must comply with data privacy laws such as the California Consumer Privacy Act (CCPA) to ensure legal access and use of users' data.

3.2. Ethical Requirements

- ❖ The Shipping Kiosk software will ensure that users are aware of exactly how their data will be used.
- ❖ No data will be sold for any purpose.
- ❖ Once no longer required for the purpose of delivering the package, the software will safely delete users' data.

3.3. Legislative Requirements

3.3.1. Accounting Requirements

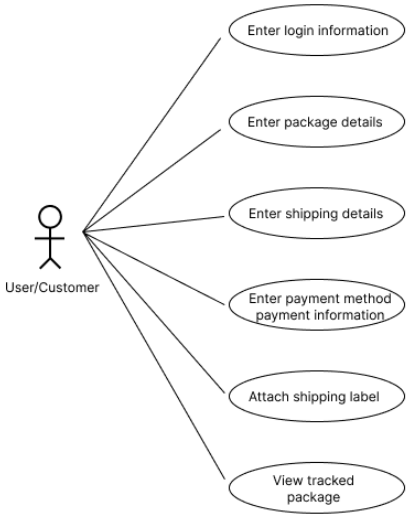
- ❖ The database must contain detailed logs for all transactions.
- ❖ Users' payment information must not be stored in the database.
- ❖ In order to scale the system globally, the system must be able to handle financial transactions with different currencies.

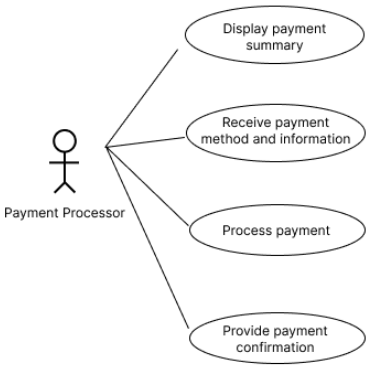
- ❖ The software must calculate shipping and packaging costs based on the users' desired packaging materials, delivery timeline, package dimensions, and item weight.

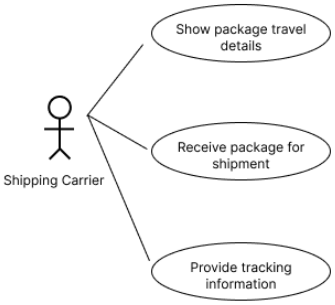
3.3.2. Safety/security Requirements

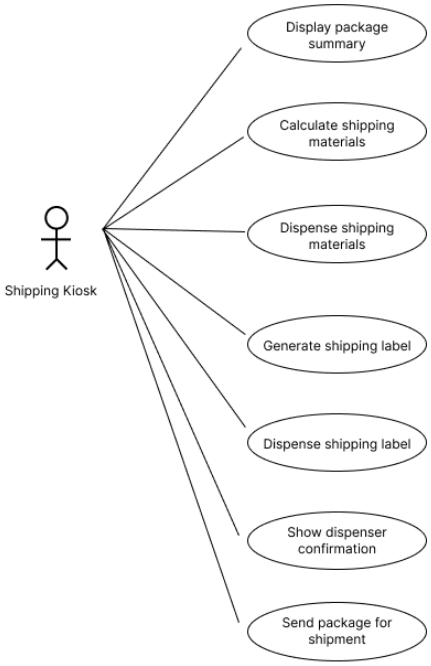
- ❖ The Shipping Kiosk system shall implement the data privacy laws of the country it is in.
- ❖ The system must encrypt all users' personal identifiable data.

Section 2.7: Use Case Diagram



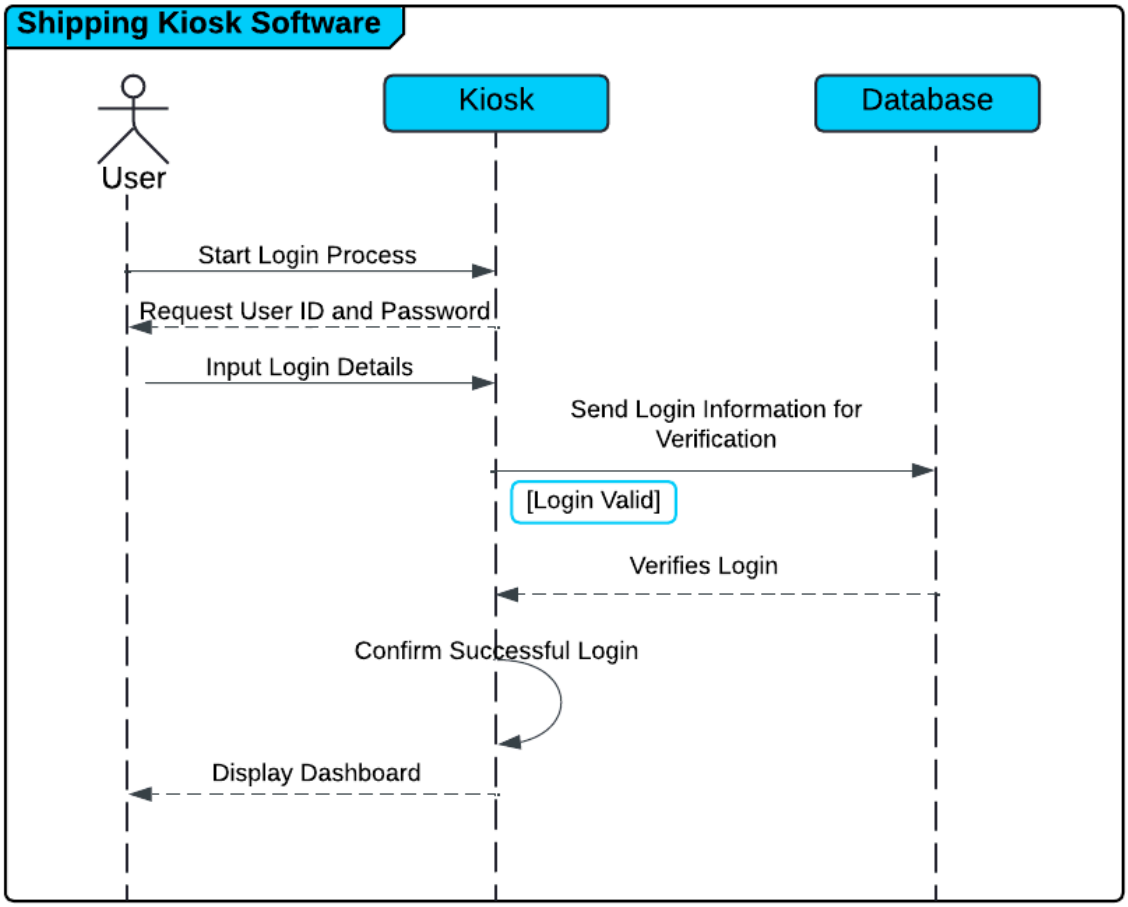




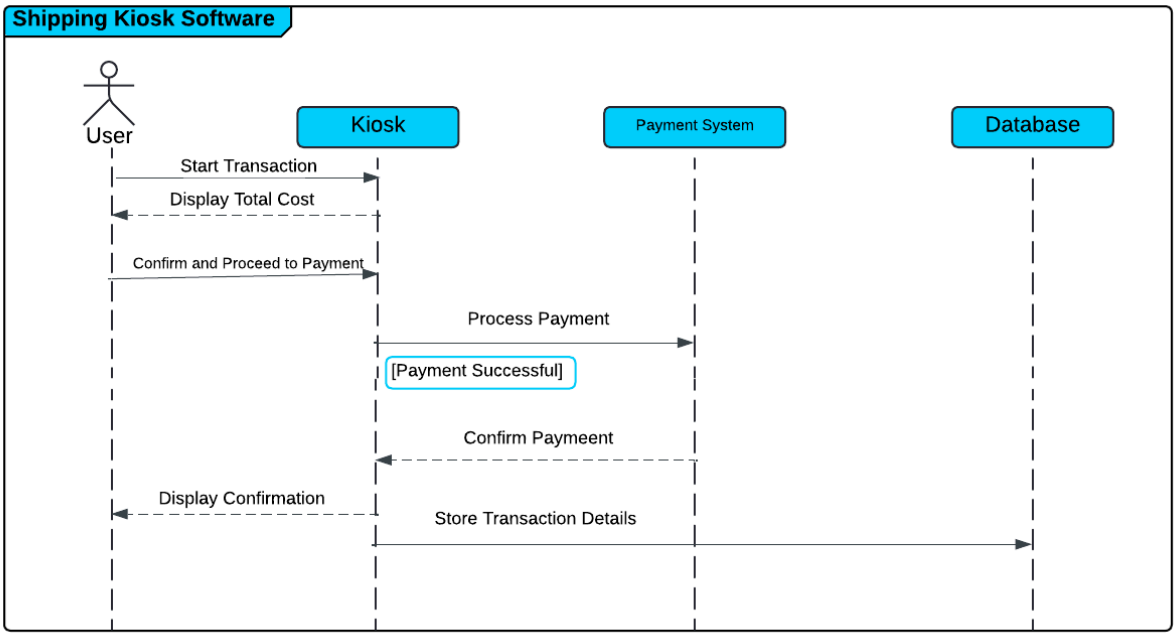


Section 2.8: Sequence diagrams

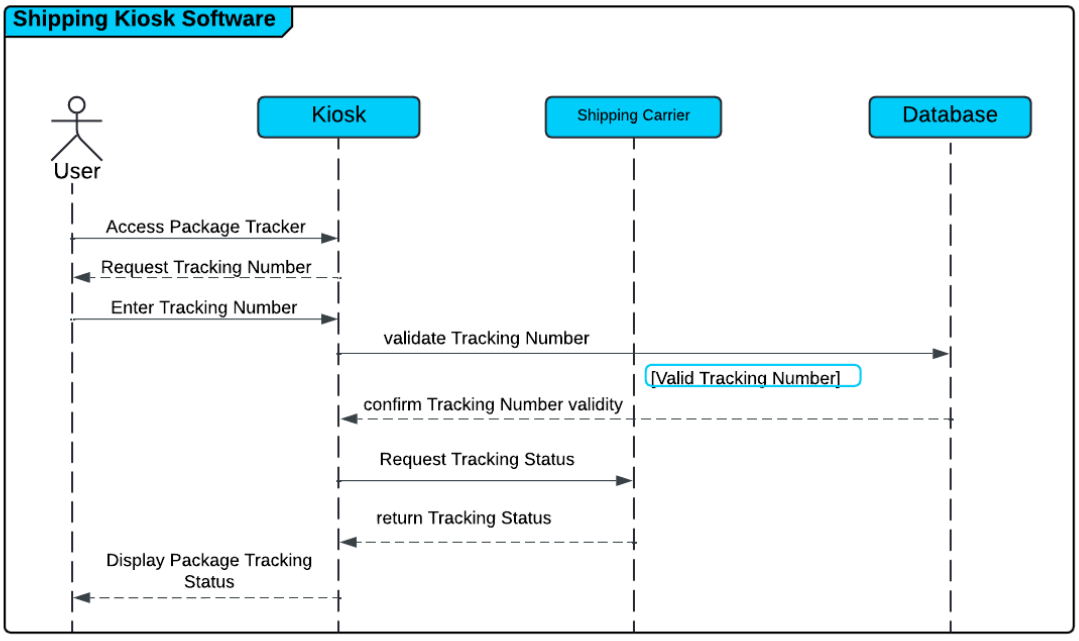
Enter Login Sequence



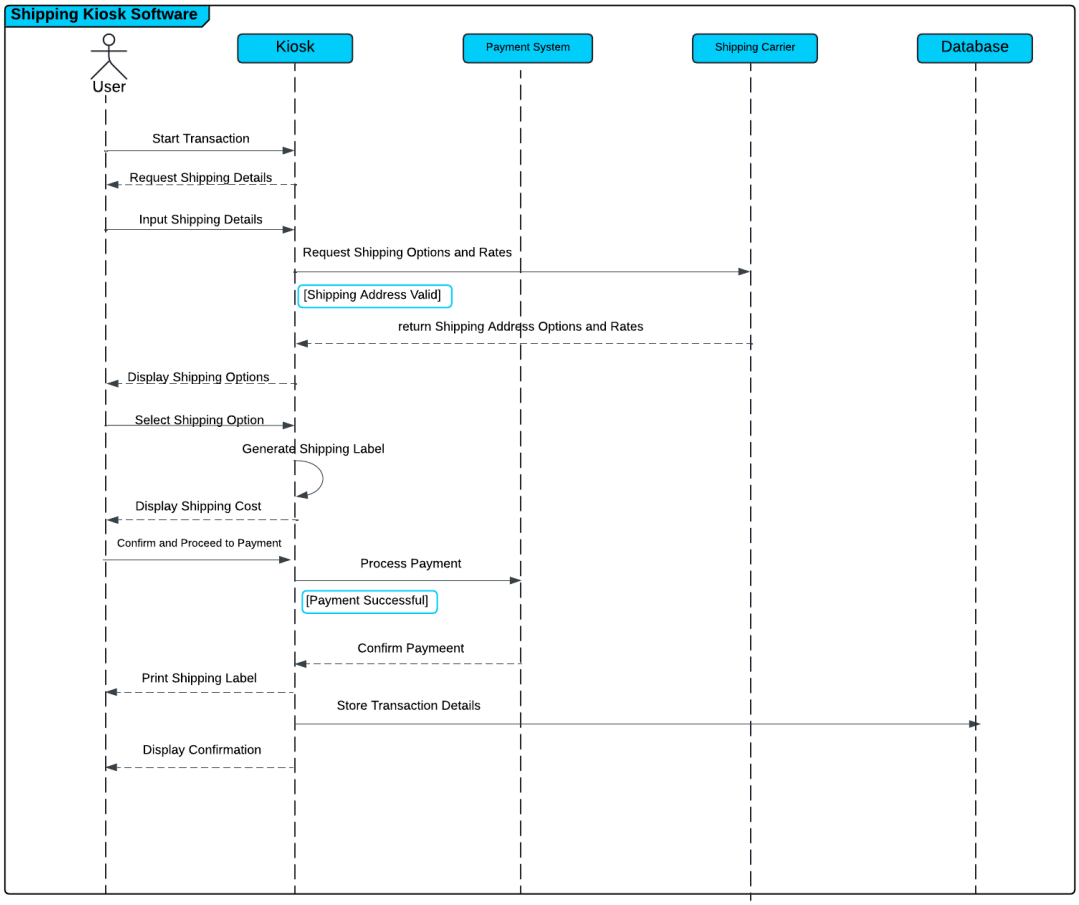
Process Payment Information

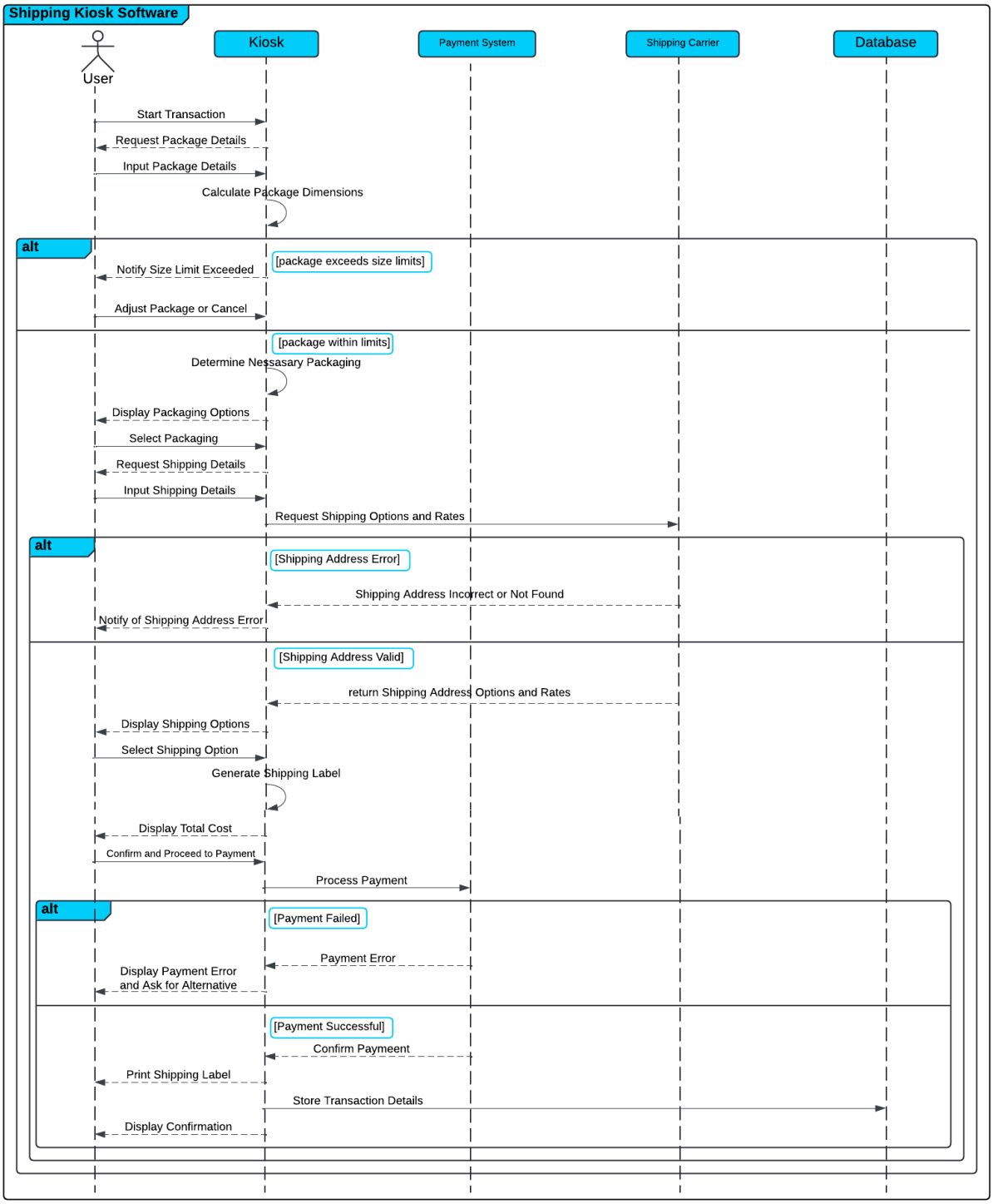


View Tracked Package

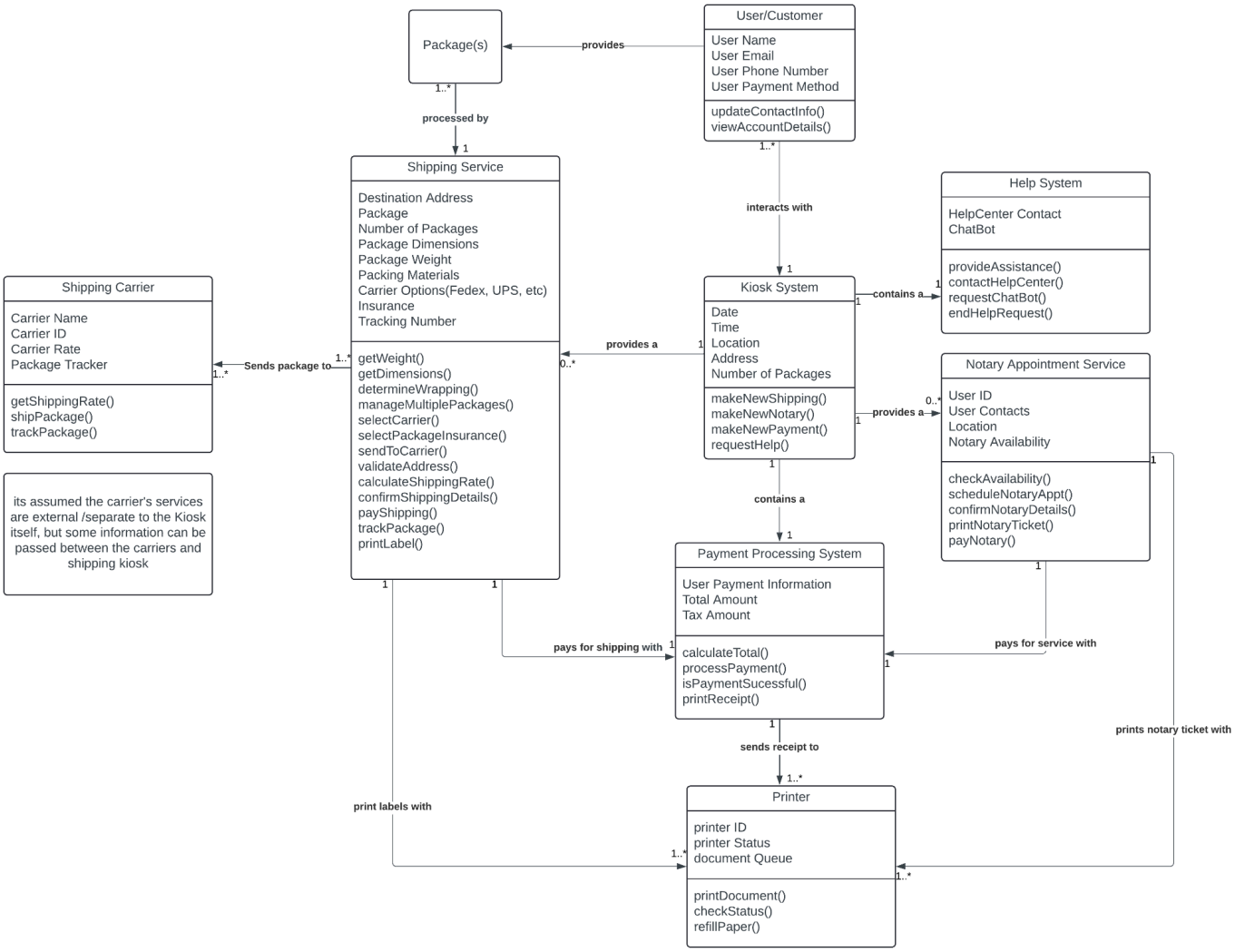


Enter Shipping Details

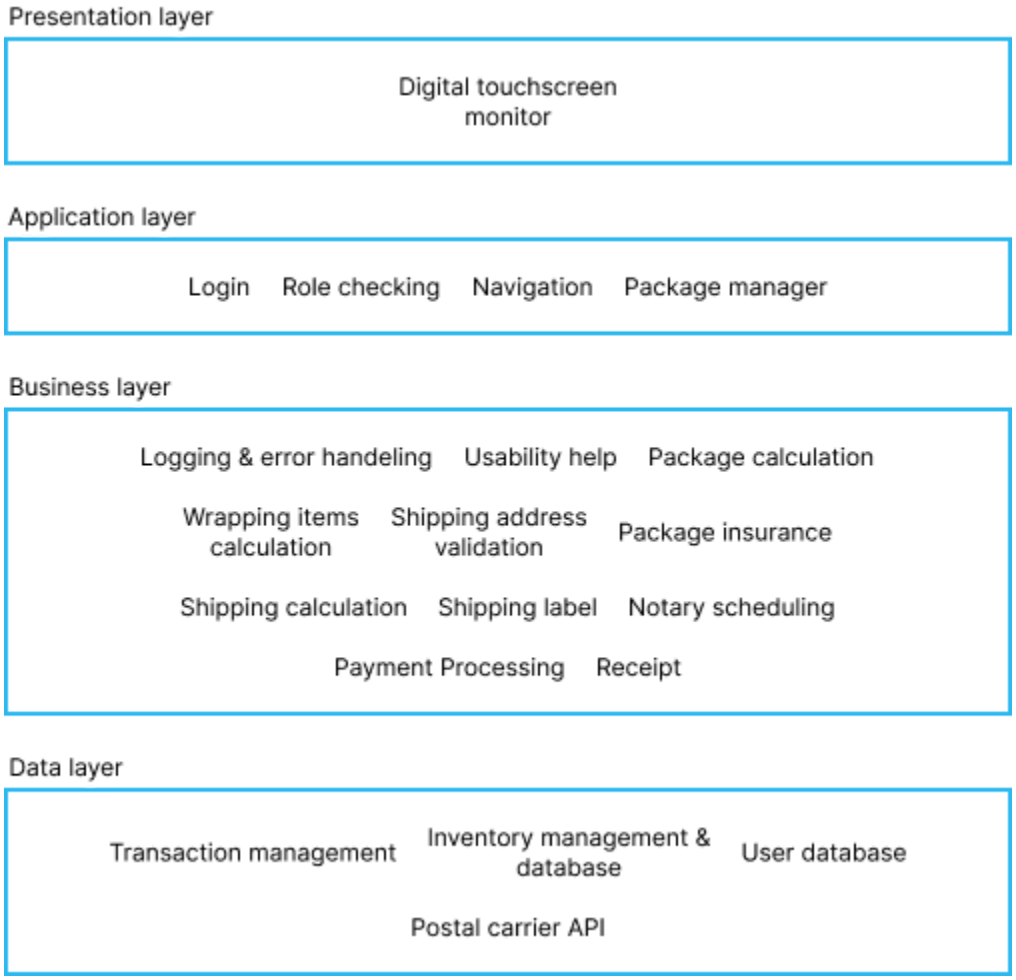



Shipping Kiosk Software
User
Kiosk
Payment System
Shipping Carrier
Database
Start Transaction
Request Package Details
Input Package Details
Calculate Package Dimensions
alt
[package exceeds size limits]
Notify Size Limit Exceeded
Adjust Package or Cancel
[package within limits]
Determine Nessasary Packaging
Display Packaging Options
Select Packaging
Request Shipping Details
Input Shipping Details
Request Shipping Options and Rates
alt
[Shipping Address Error]
Shipping Address Incorrect or Not Found
Notify of Shipping Address Error
[Shipping Address Valid]
return Shipping Address Options and Rates
Display Shipping Options
Select Shipping Option
Generate Shipping Label
Display Total Cost
Confirm and Proceed to Payment
Process Payment
alt
[Payment Failed]
Payment Error
Display Payment Error
and Ask for Alternative
[Payment Successful]
Confirm Paymeent
Print Shipping Label
Store Transaction Details
Display Confirmation

Section 2.9: Class Diagram



Section 2.10: Architectural Design



Section 3: Project Scheduling, Cost, Effort and Pricing Estimation, Project Duration, and Staffing

Section 3.1: Project Scheduling

Start Date: September 1, 2024

End Date: August 31, 2025

Duration: 12 months

*** *Excluding weekends*

Number of working hours per day for the project: 3-4 hours

Schedule details:

Task	*Sep*	*Oct*	*Nov*	*Dec*	*Jan*	*Feb*	*Mar*	*Apr*	*May*	*Jun*	*Jul*	*Aug*
Define scope and motivation for project												
Define requirements, software process model, and architectural design												
Implement prototype												
Implement software and hardware (Agile development)												
Test in several postal office locations												
Receive feedback from consumers												

Justification for estimation:

A 12-month timeline for developing the Self-Use Shipping Kiosk Software is necessary due to the ambitious scale of the project, the stringent functional and non-functional requirements, and the need to ensure data security. Robust system architecture that can manage high transaction volumes in multiple postal office locations is required for scalability. The hardware must also be dependable and easy to install and fix. Ensuring secure payment for users also requires careful, time-consuming integration with hardware. The project also needs to comply with data security regulations in different countries. 12 months is thus needed for the project to go through multiple rounds of testing, consumer feedback, and development to ensure creation of a high-quality, scalable, and reliable product.

Section 3.2: Cost, Effort and Pricing Estimation

An *algorithmic cost* model (or *Application Composition Model*) - COCOMO with Organic Modality is used to provide structured estimates based on the project size (small) and the relative complexity. The reason behind selecting this modality is based on the size of the project as well as the experience of the developers. Since the shipping kiosks' hardware come with pre-determined software, it is best to hire developers with experience with such systems who possess knowledge about integrating the kiosk features with shipping services, such as UPS, Fedex, etc. This would help reduce overhead costs associated with hiring more developers and

renting an expensive office space as now work can be done remotely by a small team made up of experienced developers. This would also allow us to meet the 12-month project development deadline as mentioned in Section 3.1.

The figures below display the projected estimate for the initial project. The *a, b, c, d* values were the pre-set values associated with the semi-detached modality [7]. The LOC Estimates are fictitious values assigned for key code components. These were then used to generate the final KLOC (thousands of lines of code) that was used to compute Effort, Development Time, Staff size and Productivity estimates. The formulae used in the computation are also listed below.

COCOMO Model Mode: Organic

a	b	c	d	KLOC
2.4	1.05	2.5	0.38	22
Effort	61.62	person-months		
Development Time	12.0	months		
Avg. Staff Size	5.15	persons		
Productivity	0.357	person-months		

KLOC Estimate	
Feature	**KLOC**
UX/UI Specifications	2
Architecture and APIs	4
Payment Processing	5
Label Printing	2
Package Tracking	3
Error Handling	3
Payment Estimate	3
Total LOC	**22**

Effort = a * (KLOC^b)
Development Time = c * (effort^d)
Avg. Staff size = effort/development time
Productivity = KLOC/Effort
Source: Software Engineering Economics - Barry Boehm (1981)

Section 3.3: Estimated Cost of Hardware Products

Hardware Component		Cost
Core Hardware	All-In-One Touchscreen/Industrial Panel PC	$2,110.00
	Power Supply	$64.00
	Metal Casing/Outer Shell	$900.00
	Server	$350.00
Necessary Peripherals	Weight Scale	$230.00
	Digital Measuring Tape	$75.00
	Keyboard	$115.00
	ADA Keypad	$250.00
	Shipping Label Printer	$93.00
	Barcode Scanner	$150.00
	Receipt Printer	$600.00
	POS Terminal	$499.00
	Imbedded Cash Recycler	$800.00
	Microphone Array Module	$242.00
	Total	**$6,478.00**

The estimated cost of the hardware products is calculated with a singular kiosk in mind. Prices were sourced by companies with listed components and prices available for commercial purchase [8]. The components consist of the main core hardware intended to create the main kiosk such as the power supply, the industrial panel computer, and the metal casing. An industrial panel PC replaces the need for finding individual computer hardware components (CPU, storage, Wi-Fi module, etc) and reduces costs. The peripheral components, those that are input/output devices, allow for the actual functionality of the kiosk to take place. Components for measurement, payment acceptance and user input are needed in order to use the components for printing, dispensing materials and money.

Additional costs incur in the form of compliance with the Americans with Disabilities Act, which ensures that people with disabilities have equal access to use of such devices [9]. Self-serve kiosks must be ADA compliant, especially if used for a business such as the proposed shipping kiosk. The integrated keyboard, NavPad, and microphone array module allow for navigation of the kiosk without the touch screen and with use of voice input, if needed [10]. With those additions, the final cost, on the higher end of estimates, came out to $6,478 for the singular kiosk unit.

Further production would likely cost less with wholesale prices or by using a kiosk designing service like KIOSK, which offers estimates from $900 to $4,000 for a custom kiosk [11].

Section 3.4: Estimated Cost of Software Products

Key Component	Cost
Requirements Engineering and Project Planning (10%)	$ 30,000.00
Architecture and UI/UX design (10%)	$ 25,000.00
Back-end code generation (40%)	$ 60,000.00
Front-end code generation (25%)	$ 45,000.00
Software Testing and Deployment (15%)	$ 10,000.00
Initial Development Cost	**$ 170,000.00**
Software Maintenance (recurring monthly)	$ 3,000.00
Updates and Support (recurring monthly)	$ 4,000.00
Cloud/Server Hosting (recurring monthly)	$ 2,000.00
Monthly Software Operational Cost	**$ 9,000.00**

Based on some secondary research, an estimate for the cost of software products were computed. These estimates are fictitious but present close to realistic figures for a small-medium size application including the type of software components, the relative proportion of their cost, as well as an estimate for monthly operational costs that a shipping kiosk would incur in maintenance and upkeep of the software [12]. These estimates are used to project an initial cost figure which will become more accurate over the course of the project as greater uncertainty is involved in the initial phases of a project (as illustrated by the Cone of Uncertainty diagram) [7][13].

Section 3.5: Estimated Cost of Personnel

Amount	**Personnel**	**Monthly Cost**
2	Software Developer	$13,333.33
1	UI/UX Designer	$5,833.33
1	Quality Assurance Engineer	$5,833.33
1	Project Manager	$8,333.33
	Initial Development Cost	33,333.32
1	Maintenance Engineer	$5,416.00
	Ongoing Maintenance Cost	$5416.00
	Training for End-User	$2,000

	Training Cost	$2,000
	Monthly Total	**$40,749.32**

After further research, the monthly cost of hiring personnel for various stages of the software engineering process was computed. Firstly, the personnel required for the initial project development was obtained, and the annual salary for each of these personnel was reported. To program and build the initial project, 2 software engineers and a UI/UX designer will be necessary to program the frontend and backend of the application. The annual salary of a software developer is about $80,000 and since there will be 2 developers, the cost would be approximately $160,000 annually. From this, the monthly cost will be approximately $13,333. The annual salary of a UI/UX designer is approximately $70,000 and the monthly cost will be about $5833. Furthermore, a quality assurance engineer will be required to validate the quality of the code. The annual salary of a quality assurance engineer is approximately $70,000 annually, so the monthly cost will be about $5,833. A project manager will also be necessary to keep the development team on track to meet deadlines and deliverables. Since the annual salary of a project manager is $100,000, the monthly cost of this personnel will be about $8,333. Thus, the total monthly cost for personnel during the initial development will be about $33,333.32.

Next, the cost for training the personnel at the post offices who will be using the shipping kiosks was calculated. Assuming that one shipping kiosk will be installed every month, the monthly cost for training local personnel (end users) will be approximately $2,000.

Finally, a maintenance engineer will be necessary to conduct periodic maintenance and reparations of the shipping kiosk application. The annual salary of a maintenance engineer is about $65,000 so the monthly cost for ongoing maintenance will be about $5,4156. After adding all the monthly expenses, it was determined that the monthly cost of personnel will be approximately $40,749.32, making the annual personnel cost about $488,991.

Section 4: Software Test Plan

a.) To verify the accuracy and reliability of the shipping label created by the kiosk, we ensure that each component of the address validation system works as intended. This includes testing individual validation methods and the complete address validation method to catch edge cases, formatting issues, and invalid data. The test plan covers Name, Address, Apt/Suite, City, State Code, Zip Code, and all combined. This is a functional and black-box testing approach, focusing on verifying the expected output based on known input values without consideration for internal implementation details. For this testing, we decided to use JUnit 5 for automated unit tests and assertions such as assertEquals, assertTrue, and assertFalse to validate results. The test cases for the ShippingLabel software make sure every part of the address is correctly formatted and meets basic requirements. For name, we check that it’s not empty or just spaces. Address needs to have

both numbers and letters (ex.“123 Main St”). Apt/Suite can be empty, or it can have numbers and letters but no special characters. The city should only have letters and spaces, and the state code has to be a valid two-letter abbreviation like “TX” or “CA.” Zip code should be either five or nine digits. Finally, the complete address test makes sure that all these parts together create a properly formatted address. If even one part is wrong, the whole address is incorrect. For test execution, we decided to run unit tests and test suite execution during our testing process and expected each test case to pass. However, if we came across any failing test cases we resolved them right away.

b & c.)
isNameValid Method

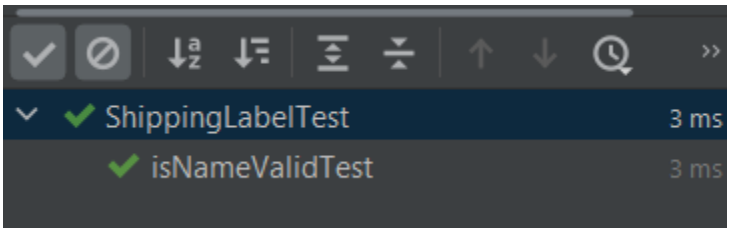
```
//In order for the name to be valid, it cannot be empty
6 usages
public static boolean isNameValid(String name) {
    return name != null && !name.trim().isEmpty();
}
```

isNameValidTest Method

```
@Test
public void isNameValidTest() {
    // Valid cases
    assertEquals( expected: true, ShippingLabel.isNameValid("Nivedha Sreenivasan"));
    assertEquals( expected: true, ShippingLabel.isNameValid("John Doe"));

    // Invalid cases
    assertEquals( expected: false, ShippingLabel.isNameValid("   "));
    assertEquals( expected: false, ShippingLabel.isNameValid(""));
    assertEquals( expected: false, ShippingLabel.isNameValid(null));
}
```

isNameValidTest Results



isAddressValid Method

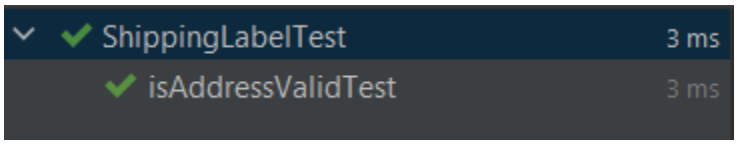
```java
//In order for the address to be valid, it cannot be empty and can only contain numbers and letters
6 usages
public static boolean isAddressValid(String address) {
    return address != null && address.matches( regex: "^[0-9]+\\s+[a-zA-Z\\s]+$");
}
```

isAddressValidTest Method

```java
@Test
public void isAddressValidTest() {
    // Valid cases
    assertEquals( expected: true, ShippingLabel.isAddressValid("123 Main St"));
    assertEquals( expected: true, ShippingLabel.isAddressValid("456 Elm Avenue"));

    // Invalid cases
    assertEquals( expected: false, ShippingLabel.isAddressValid("Main St"));
    assertEquals( expected: false, ShippingLabel.isAddressValid("123"));
    assertEquals( expected: false, ShippingLabel.isAddressValid(""));
}
```

isAddressValidTest Results



isAptSuiteValid Method

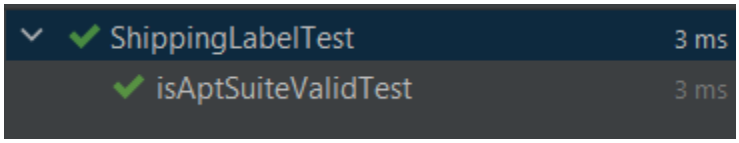
```java
// Apt/Suite is valid if it's null, empty after trimming, or contains only numbers and letters
7 usages
public static boolean isAptSuiteValid(String aptSuite) {
    return aptSuite == null || aptSuite.trim().isEmpty() || aptSuite.trim().matches( regex: "[a-zA-Z0-9\\s]+");
}
```

isAptSuiteValidTest Method

```
no usages
@Test
public void isAptSuiteValidTest() {
    // Valid cases
    assertEquals( expected: true, ShippingLabel.isAptSuiteValid(""));
    assertEquals( expected: true, ShippingLabel.isAptSuiteValid("101"));
    assertEquals( expected: true, ShippingLabel.isAptSuiteValid("Apt 4B"));
    assertEquals( expected: true, ShippingLabel.isAptSuiteValid("Suite 101"));

    // Invalid cases
    assertEquals( expected: false, ShippingLabel.isAptSuiteValid("Apt#5"));
    assertEquals( expected: false, ShippingLabel.isAptSuiteValid("4B!"));
}
```

isAptSuiteValidTest Results



isCityValid Method

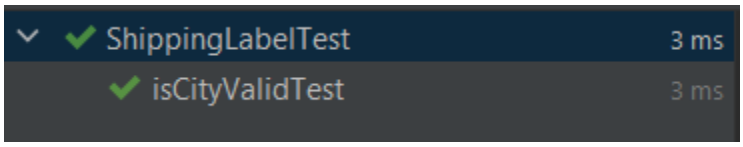
```
//In order for the city to be valid, it cannot be empty and can only contain letters
10 usages
public static boolean isCityValid(String city) {
    return city != null && city.matches( regex: "^[a-zA-Z\\s]+$");
}
```

isCityValidTest Method

```
no usages
@Test
public void isCityValidTest() {
    // Valid cases
    assertEquals( expected: true, ShippingLabel.isCityValid("Dallas"));
    assertEquals( expected: true, ShippingLabel.isCityValid("San Francisco"));
    assertEquals( expected: true, ShippingLabel.isCityValid("New York"));
    assertEquals( expected: true, ShippingLabel.isCityValid("Los Angeles"));

    // Invalid cases
    assertEquals( expected: false, ShippingLabel.isCityValid("Dallas123"));
    assertEquals( expected: false, ShippingLabel.isCityValid("New-York"));
    assertEquals( expected: false, ShippingLabel.isCityValid("San@Francisco"));
    assertEquals( expected: false, ShippingLabel.isCityValid(""));
    assertEquals( expected: false, ShippingLabel.isCityValid("12345"));
}
```

isCityValidTest Results



isStateCodeValid Method

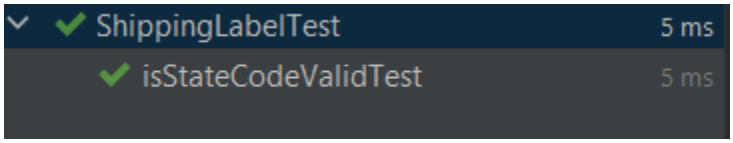
```
public static boolean isStateCodeValid(String state) {
    Set<String> stateCodes = Set.of("AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA",
            "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD",
            "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ",
            "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC",
            "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY");
    return stateCodes.contains(state);
}
```

isStateCodeValidTest Method

```java
@Test
public void isStateCodeValidTest() {
    // Valid cases (assuming valid U.S. state codes)
    assertEquals( expected: true, ShippingLabel.isStateCodeValid("TX"));
    assertEquals( expected: true, ShippingLabel.isStateCodeValid("CA"));
    assertEquals( expected: true, ShippingLabel.isStateCodeValid("NY"));
    assertEquals( expected: true, ShippingLabel.isStateCodeValid("FL"));

    // Invalid cases
    assertEquals( expected: false, ShippingLabel.isStateCodeValid("XX"));
    assertEquals( expected: false, ShippingLabel.isStateCodeValid("abc"));
    assertEquals( expected: false, ShippingLabel.isStateCodeValid("123"));
    assertEquals( expected: false, ShippingLabel.isStateCodeValid(""));
}
```

isStateCodeValidTest Results



isZipCodeValid Method

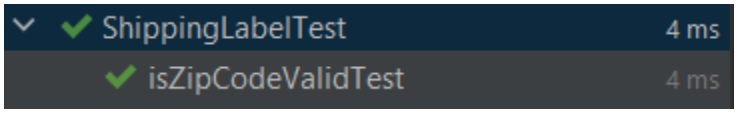
```java
// Validates a standard 5-digit or 9-digit zip code format ex. "12345" or "12345-6789"
8 usages
public static boolean isZipCodeValid(String zipCode) {
    return zipCode != null && zipCode.matches( regex: "\\d{5}(-\\d{4})?");
}
```

isZipCodeValidTest Method

```java
@Test
public void isZipCodeValidTest() {
    // Valid cases (5-digit and 9-digit formats)
    assertEquals( expected: true, ShippingLabel.isZipCodeValid("12345"));
    assertEquals( expected: true, ShippingLabel.isZipCodeValid("12345-6789"));

    // Invalid cases
    assertEquals( expected: false, ShippingLabel.isZipCodeValid("1234"));
    assertEquals( expected: false, ShippingLabel.isZipCodeValid("123456"));
    assertEquals( expected: false, ShippingLabel.isZipCodeValid("12345-678"));
    assertEquals( expected: false, ShippingLabel.isZipCodeValid(""));
    assertEquals( expected: false, ShippingLabel.isZipCodeValid("ABCDE"));
}
```

isZipCodeValidTest Results



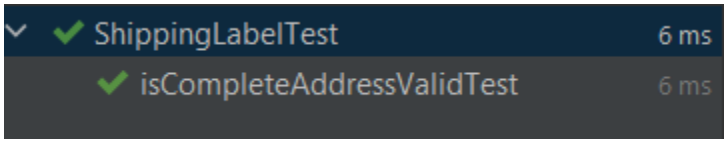
isCompleteAddressValid Method

```
// Combined validation for the entire address
6 usages
public static boolean isCompleteAddressValid(String name, String address, String aptSuite, String city, String state, String zipCode) {
    return isNameValid(name) &&
            isAddressValid(address) &&
            isAptSuiteValid(aptSuite) &&
            isCityValid(city) &&
            isStateCodeValid(state) &&
            isZipCodeValid(zipCode);
}
```

isCompleteAddressValidTest Method

```
@Test
public void isCompleteAddressValidTest() {
    // Valid complete address
    assertEquals( expected: true, ShippingLabel.isCompleteAddressValid(
                    name: "John Doe", address: "123 Main St", aptSuite: "Apt 4B", city: "Dallas", state: "TX", zipCode: "12345"));
    // Invalid cases
    assertEquals( expected: false, ShippingLabel.isCompleteAddressValid(
                    name: "  ", address: "123 Main St", aptSuite: "Apt 4B", city: "Dallas", state: "TX", zipCode: "12345"));
    assertEquals( expected: false, ShippingLabel.isCompleteAddressValid(
                    name: "John Doe", address: "Main St", aptSuite: "Apt 4B", city: "Dallas", state: "TX", zipCode: "12345"));
    assertEquals( expected: false, ShippingLabel.isCompleteAddressValid(
                    name: "John Doe", address: "123 Main St", aptSuite: "Apt 4B", city: "Dallas123", state: "TX", zipCode: "12345"));
    assertEquals( expected: false, ShippingLabel.isCompleteAddressValid(
                    name: "John Doe", address: "123 Main St", aptSuite: "Apt 4B", city: "Dallas", state: "XX", zipCode: "12345"));
    assertEquals( expected: false, ShippingLabel.isCompleteAddressValid(
                    name: "John Doe", address: "123 Main St", aptSuite: "Apt 4B", city: "Dallas", state: "TX", zipCode: "1234"));
}
```

isCompleteAddressValidTest Results



Sections 5: Software Comparison

After looking into designs that are similar to our idea of a Shipping Kiosk, the three similar designs that we discovered are the USPS Self-Service Kiosks, FedEx Ship & Go Kiosks, and UPS Self-Service Terminals.

Some of the features that the USPS kiosks allow are for customers to weigh packages, calculate postage, print shipping labels, and pay for postage. Each of these features is similar to the features that our proposed Shipping Kiosk boasts. However, some of the key differences are that the USPS Self-Service Kiosks place a few different limitations on their customers, the most important of which are not providing integration with advanced packaging options or item wrapping services. The model that we propose improves on the existing USPS kiosks by including automatic package dimension measurement, dispensing wrapping materials, and handling multiple packages at once. With our improved model, we are taking into account the needs of the user and giving them a more streamlined experience with our kiosk. We are not allowing any user error in things such as the measurement of the package, and are able to automate the process by calculating those values as a part of the features that the Shipping Kiosk includes [4].

The second shipping kiosk design that is similar to our design is the FedEx Ship & Go Kiosks. Some of the features that the FedEx Ship & Go Kiosks offer are options for printing labels, shipping packages, and checking shipping rates. Some of the key differences between the FedEx Ship & Go Kiosks and our proposed Shipping Kiosk are that the FedEx Ship & Go Kiosks do not offer notary services or integrated packaging services. Our Shipping Kiosk offers integrated notary scheduling, suggests packaging items, and also provides additional safety features post-COVID-19. This is an improvement on the existing model because we are able to address safety concerns that users may have while also making the process a little more automated and trying to reduce user error [5].

The final shipping kiosk design that we found that is similar to our design is the UPS Self-Service Terminals. These terminals are mainly used for printing shipping labels and processing payments. Some of the key differences between our Shipping Kiosk and the UPS Self-Service Terminals are that the UPS Self-Service Terminals lack automated packaging assistance or tools to manage packages simultaneously. We were able to improve upon this design by providing a more comprehensive design that showcases an automated wrapping dispenser and bulk package management. Again, this focuses on the user experience and allows them to have a more streamlined experience with reduced margins for error [4].

Section 6: Conclusion

Section 2.8 critique required for multiple individual use cases to be made, and was a bit challenging as it was a bit time consuming to think of all the interactions that could happen in each use case. Section 2.9 had to be redone as some of the methods and attributes were redundant, but the changes brought some insight into how to improve the efficiency of each of the methods.

Section 3.1 was challenging since it is difficult to estimate the time required to develop a project. I focused on defining distinct stages in the timeline and overlapped certain stages to align with the core Agile principles. Development time varies with team expertise, number of iterative cycles, consumer feedback, and test success. Thus, I estimated that development will take around 8 months.

Section 3.2 was a bit of a challenge and required a do-over as the effort and development time calculations had to match the application's size as well as the projected timeline of 12 months justified in Section 3.1. Initially, the KLOC generated was too large (>2000) and greatly skewed the development timeline to 76 months. The KLOC was then revised to suit that of a smaller size project that allowed the development time to fit within a reasonable 12-month time frame.

Section 3.5 was a little challenging because it required careful consideration of what staff would be necessary to complete each stage of the software engineering process. It was important to optimize staff selection to minimize the cost while maximizing production ability. It was also necessary to select how many developers and designers would be necessary in order to adhere to the project timeline, as having more engineers would expedite programming.

Section 4 was difficult, as I chose to validate the address without having an API to validate the actual existence of the address. I could not use tan API due to the costs associated with it. Instead, I used validation techniques such as checking if certain values had numbers, letters, or were empty.

Section 7: References

[1] B. Wolford, "What is GDPR, the EU's new data protection law?," *GDPR.eu*. https://gdpr.eu/what-is-gdpr/

[2] FedEx, "Shipping Locations & Kiosks." [Online]. Available: https://www.fedex.com

[3] State of California Department of Justice, "California Consumer Privacy Act (CCPA)," *State of California - Department of Justice - Office of the Attorney General*, Mar. 13, 2024. https://oag.ca.gov/privacy/ccpa

[4] United Parcel Service, “UPS Self-Service Options.” [Online]. Available: https://www.ups.com

[5] United States Postal Service, “Self-Service Kiosks.” [Online]. Available: https://www.usps.com

[7] B. W. Boehm, *Software engineering economics*. Englewood Cliffs, N.J.: Prentice-Hall, Ca, 1981. Available: https://staff.emu.edu.tr/alexanderchefranov/Documents/CMPE412/Boehm1981%20COCOMO.pdf

[8] Kiosk Marketplace, "Kiosk Hardware / Kiosk Components." [Online]. https://www.kioskmarketplace.com/blogs/how-to-build-your-digital-signage-ecosystem-top-10-faq/

[9] Kiosk Group, "Ensuring Your Kiosks are ADA Compliant." [Online]. https://www.kioskgroup.com/blogs/resource-guides/ensuring-your-kiosks-are-ada-compliant

[10] Storm Interface, "Assistive Technology Products." [Online]. https://www.storm-interface.com/storm-atp-product-range

[11] Kiosk, "Custom Kiosk Design." [Online]. https://kiosk.com/custom-kiosks/

[12] ScienceSoft, “Software Development Costs: How to Estimate and Reduce,” *www.scnsoft.com*. https://www.scnsoft.com/software-development/costs

[13] “Software Cost Estimation Explained,” *SEI Blog*, June 17, 2024. https://insights.sei.cmu.edu/blog/software-cost-estimation-explained/