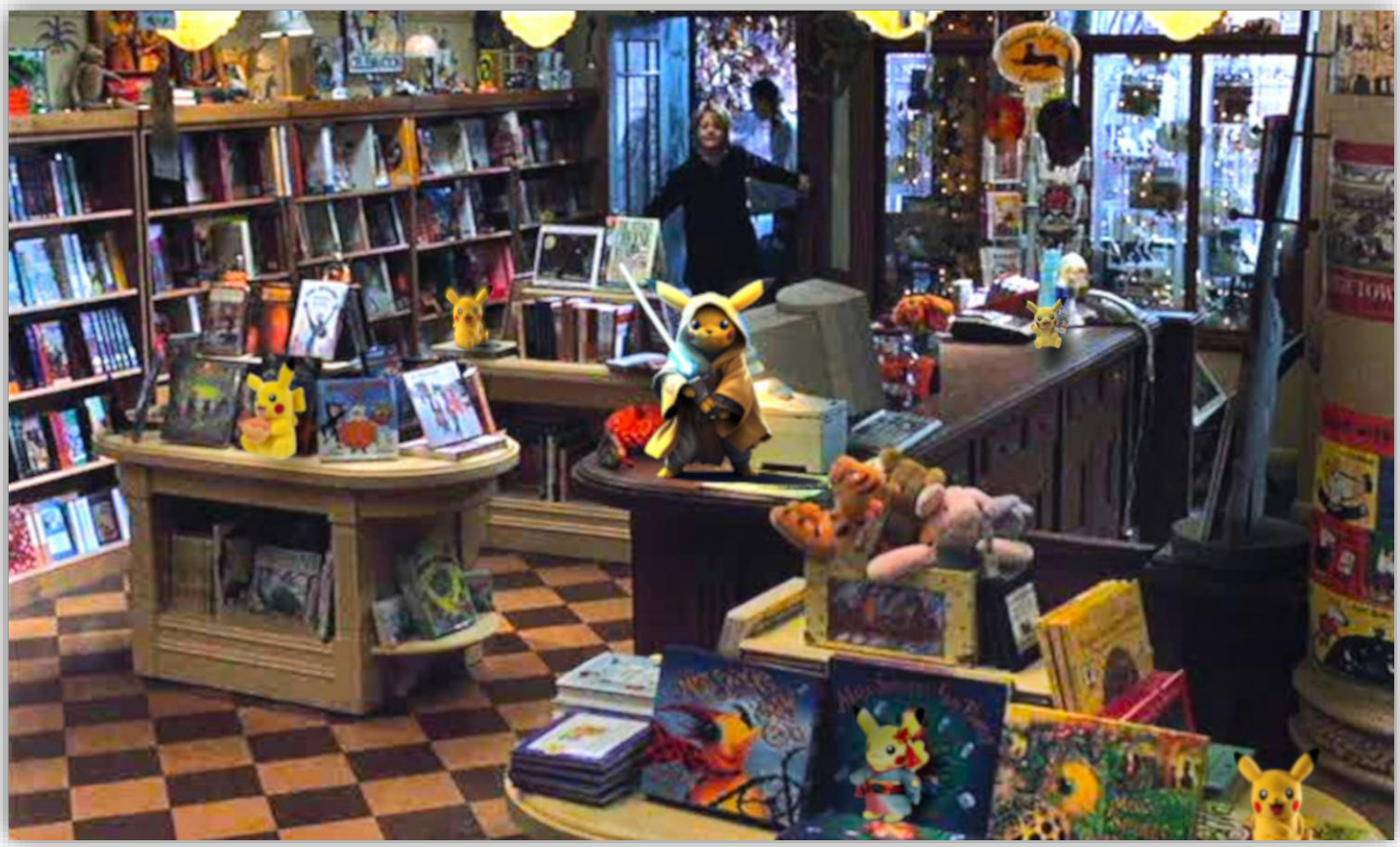
Juan Antonio Rojas
CEN 3024C
15339
Sept. 16, 2024

Requirements Definition Planning Document



Pikazon, Independent Booksellers

Index

1. Introduction

"The goal of requirements engineering is to understand what various stakeholders need from a system, and to document these needs in a way that they can be analyzed, communicated, and agreed upon." (Ian Sommerville, Chapter 4, p. 83)

1.1 Purpose

This document aims at defining the software requirements for an eCommerce system for Pikazon (a portmanteau of the fictitious character **Pika**chu and the ecommerce giant Ama**zon**). This system will support both the charming brick-and-mortar store (the shop around the corner from 106 West 69th Street, NY, NY) as well as online sales, constantly alerting us: "You've got mail."
This requirements document will serve as a guide for our development team with the understanding that all aspects of the system are well-covered and agreed upon. This eCommerce system will expand **Pikazon's** ability to sell products to the online community by integrating a friendly UI, payment processing inventory management, and distribution logistics.

1.2 Scope

We will look at the expected functional and non-functional requirements for our little online bookstore, including casual browsing, book searches, customer registration, placing orders, handling payments, inventory management, order fulfillment and delivery, customer support, etc.

Our system will follow a 3-tier architecture, dividing the application into the **presentation** layer, **application logic** layer, and **data** layer. Each tier will be robust enough to handle peak usage and also allow for future growth.

1.3 Glossary

It's important to know some technical definitions, acronyms, and abbreviations to have a shared understanding of key terms and concepts related to the project

Term	Definition
API	Application Programming Interface, which allows interaction between software components.
3-tier architecture	A system architecture that separates the application into three logical layers: presentation, application, and data.
1. Presentation Layer	The user interface or front-end part of the system that customers interact with (e.g., web browser, mobile app).
2. Application Layer	The middle layer where the business logic and processes are executed (e.g., processing orders, payment transactions).
3. Data Layer	The back end where the database resides, handling data management, retrieval, updates, and deletes.
Database Schema	The structure of the database, including tables, fields, relationships, and constraints, used to store and organize data.
CRUD	Create, Read, Update, Delete; the four basic operations for managing data in a database.
Payment Gateway	A service that processes online payments securely.

Scalability	The ability of a system to handle increased load or demand by adding resources.
POS	Point-of-Sale, the system used to process in-store purchases, like the old cash register.
OMS	Order Management System, used to track and manage customer orders, including processing, shipping, and returns. Like our new software package.
SKU	Stock Keeping Unit, a unique identifier for each product in inventory, popularized by Walmart, Kmart, Sears and JCPenney back in the early 1970s.
ISBN	The **I**nternational **S**tandard **B**ook **N**umber purchased by a publisher from an affiliate of the International ISBN Agency. A different ISBN is assigned to each separate edition and variation of a publication, but not to a simple reprinting of an existing item. For example, an e-book, a paperback and a hardcover edition of the same book must each have a different ISBN, but an unchanged reprint of the hardcover edition keeps the same ISBN. (https://en.wikipedia.org/wiki/ISBN). Pronounced “isbun” or “isben.”
UI	User Interface. The design and layout of the website that users interact with, including buttons, forms, and navigation.
UX	User Experience. The overall experience and satisfaction of users when interacting with the website, encompassing UI design, ease of navigation, and accessibility.
Responsive Design	An approach to web design that ensures the website functions well on various devices and screen sizes, including desktops, tablets, and smartphones.
eCommerce	The software framework used to build and manage the online store, including features like product management, payment processing, and order fulfillment.
CMS	Content Management System. Software used to manage and update the content of the website (WordPress, Magento, etc.).
CDN	Content Delivery Network. A network of servers that distributes content to users based on their geographic location, improving load times and performance.
CRM	Customer Relationship Management. A system for managing interactions with current and potential customers, including tracking customer behavior and managing communications.
SEO	Search Engine Optimization. The practice of improving the website’s visibility in search engine results to attract more organic traffic.
XML	e**X**tensible **M**arkup **L**anguage. A markup language used for storing and transporting data. Often used for data interchange between systems.
SSL	Secure Sockets Layer. A security protocol that encrypts data transmitted between the user's browser and the server, ensuring secure transactions.
Spring Boot	A 2014 framework invented at Pivotal Software (now part of VMware), made to simplify the setup and development of Java applications by providing production-ready defaults and reducing configuration overhead. It can be integrated with CDNs to optimize the delivery of static content in applications.
TB (WD)	Tera Byte = 1024 Giga Byes or 1 million Mega Bytes. WD id the Western Digital brand
RAID 5	Redundant Array of Independent Disks. RAID 5 uses striping and parity, but it does not use mirroring. That can be accomplished with a NAS
NAS	Network-Attached Storage that connects to a network, allowing multiple devices to access and share the storage over the network. It can be configured to have mirroring.

2. System Overview

"Always deliver more than expected." (Larry Page)

2.1 Business Objectives

We are a small local bookstore seeking to expand to the online community at large with this eCommerce website and deliveries through USPS/UPS/FedEx. But we're not going all out on this, we want to keep our mom-and-pop feel. We're just opening a window to the wide world out there. We will integrate our small cash register system with our new shopping cart, whereby each in-store sale will be made through a countertop tablet with a CC reader linked to a payment gateway, which in turn will be linked to our bank account.

There are several readers to choose from, such as SwipeSimple, Square Terminal, SumUp, Helcim, Ingenico, Pax, etc., and also PayPal or Shopify card readers. In the future we might adopt a more advanced POS system from Square, Clover, Vend, Lightspeed, SumUp, Epos Now, Shopify or even Amazon (most of which offer a complete ecommerce packages), but for now we'd like to try this simple custom ecommerce system due to our budgetary constraints.

2.2 System Objectives

"Design is not just what it looks like and feels like. Design is how it works." (Steve Jobs)

This software should be simple and easy to use, for both online customers and employees. It is connected to an inventory database that allows for book searches and selections for purchase (through a shopping cart), with real-time inventory updates on every sale (online as well as in-store). The system will allow the online user to check on his or her delivery, to provide reviews about the books, as well as receive recommendations for purchasing similar items. It also allows the store manager to place orders and track incoming inventory.

It must be noted that we depend on third-party services such as a payment gateway, shipping services, drop-shipping, and our book suppliers. So, our system's integration with them is a must.

2.3 Target Audience

First we need to make sure our **staff** is familiar with it. They need to understand all the nuances of the system, as a 5th grader would, i.e. people with no technical expertise, including the **manager**.

The system will be so intuitive and user-friendly that our **online customers** will enjoy the experience of shopping with us, as if they were at the store. In the future, we will promote through social media, but for now our marketing will be done just by word of mouth, with the help of occasional events such as monthly book reviews and author's signatures, or weekly children storytelling, etc. We believe that by improving customer engagement, we will create a delightful shopping experience and therefore increase sales.

The application will support both English and Spanish languages and be adaptable to different currencies.

3. Functional Requirements

"Functional requirements define the interactions between the system and its users or other systems. They specify what the system should do and how it should respond to inputs, providing a detailed description of the system's behavior." (Ian Sommerville, Chapter 4, p. 85)

3.1 User Types and Roles

3.1.1 Online Customer: Anybody with an internet connection has the option to become a store customer (set up an account with username and password, and email for confirmation)

3.1.2 Admin/Manager: In charge of ordering new books from publishers, updating the backend database (with the title, author, description, quantity on hand, etc.), and the front-end catalog (pictures, description, price, etc.). They will keep an eye out on stock levels (reorder if necessary), monitor customer's orders and handle returns/refunds, as well as respond to their comments and book reviews. They can also manage promotions, coupons, and discounts, as well as events, for better PR.

3.1.3 In-store Employee: they should be savvy on cash handling (give the proper change, etc.), but also intuitive enough to learn the eCommerce system and process a sale just as if they were an online customer (login, find item, add to cart, check out, and hand deliver the book). They must be service oriented to politely assist every customer that walks in the door (even if it's Julia Roberts).

3.2 Account Management

3.2.1 Registration: Anybody interested in signing up will be asked to provide a small profile with his/her name and email address, as well as set up a username and password. They will be given the option (check box) to receive occasional emails about promotions, events, discounts, coupons, etc. Online buyers can also opt to have two-factor authentication (2FA), but a physical address and credit card information will be required. We will implement strict validation checks in our front-end forms, as well as in our back-end application layer. To us security is paramount.

3.2.2 Account Management: Customers can update the aforementioned profile at any time. They can change username, password, email, picture, etc. If a customer loses his/her password, it can be recovered (or set up a new one) through their email. One of the options in their profile is their "Order History." Another couple of options in their account is a Wishlist and/or list of Favorite Books as well as their Book Reviews and Ratings.

3.3 Inventory Management

"What gets measured gets managed." (Peter Druker)

The most important aspect of this system, I cannot emphasize it enough, is the **inventory database**. Just as the saying goes "if you fail to plan, you plan to fail," we can say "if you neglect your inventory, you neglect your customers" and we don't want that.

Some of the highly regarded inventory management software options we have are:

1. **QuickBooks** Commerce: Inventory tracking, order management, multi-channel selling, and integration with accounting software. Plans start around $39/month for basic features, with advanced plans reaching up to $299/month depending on the size of the business.

2. **NetSuite** ERP: Comprehensive inventory management, financials, CRM, and eCommerce integration. Costs start at around $999/month
3. **Zoho** Inventory: Inventory tracking, order management, shipping integration, and multi-channel selling. Plans start at $0 for basic features, with paid plans starting at $39/month and going up to $299/month for more features and higher limits.
4. **Odoo**: Modular system with inventory management, sales, CRM, accounting, and more. Offers flexibility to add or remove modules as needed. This one is open source (free). The paid Enterprise version starts around $24/user/month, with additional costs for specific modules and support.

Our custom inventory database will focus on:

3.3.1 Products: Update the database with information about each product, including title, author, description, customer reviews, price, discounts and images.

3.3.1 Inventory Tracking: Update the database when books are sold, returned, or received in the back. Also, constantly check stock levels and reorder if necessary.

3.4 Online Shopping

"The goal of interactive design is to provide users with the necessary tools to accomplish their tasks in a way that feels natural and efficient." (Shneiderman & Plaisant)

3.4.1 Search: Have a search box, where users can search and filter by title, author, ISBN, etc.

3.4.2 View the book, including its description, readers' reviews, etc. Also, the book's availability or if it's in back order.

3.4.3 Add to cart: Add or remove a book to or from the shopping cart. Adjust quantities (+ or -).

3.5 Checking out

3.5.1 Payment Options: Ask customers how they would like to pay: PayPal, Apple/Google Pay, or with a Debit/Credit Card, to be processed by a payment gateway such as Stripe, Authorize.net, Square, etc. Setup an SSL connection. By the way, tax calculation is also presented.

3.5.2 Coupons: Ask customers for Coupon Codes or Discount Offers that they might qualify for. Also give the customers accumulated points for future discounts, similar to sky miles.

3.5.3 Shipping: Ask customers where to ship the book (pre-populate this field with the customer's address and a checkbox to accept it. If not checked type the new address). Give them shipping options and offer an estimated delivery date. Provide a connection to USPS/UPS/FedEx for order tracking. Users will receive notifications (via email or SMS) about delivery updates.

3.5.4 In-Store Sales: All staff members should be familiar with the ecommerce system so that when helping a customer check out, they will just use an over-the-counter tablet to finish the checkout process. This tablet will have a credit/debit card reader attached to its side. Customers could also pay in cash (so a tilt will still be required). Of course, in this case there will be no shipping since the customer will take the book at the store, but the database will be updated all the same.

3.6 Customer Support

"We see our customers as invited guests to a party, and we are the hosts. It's our job every day to make every important aspect of the customer experience a little bit better." (Jeff Bezos)

3.6.1 Channels: Customers can contact the store by phone, email, or SMS chat.

3.6.2 FAQ: We will maintain a web page with Frequently Asked Questions, with a JS accordion system for titles and subtitles on every possible issue.

3.6.3 Return Policy: Postings stating that customers can return books within 30 days will be at checkout, on the email/receipt, as well as on the countertop. Of course, the book must remain in mint condition for a full refund. The refund will take "x" number of days to process.

---

4. Non-Functional Requirements

"Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless." (Ian Sommerville, Chapter 4, p. 96)

4.1 Performance and Scalability

"Scalability isn't about optimizing for a single server; it's about distributing the workload across multiple machines while keeping the system performant and responsive." (Martin Kleppmann)

4.1.1 Page Load Time: Our system is set up to load web pages quickly, in less than 2 secs for first-time users (without cached pages) or mere milliseconds for our regular customers. These times might slightly increase due to peak demand, during the holidays for instance. At the moment we're being served by a local webhost, but when we upgrade to AWS, Google or Azure, these times should stay consistent or even go down, since the APIs and other backend services will respond instantaneously. Our goal is to ensure a smooth user interaction at all times.

4.1.2 Concurrency: The same thing applies to multiple user support. Initially, we expect at most 500 concurrent users without performance degradation. But as we grow, Amazon, Google or Microsoft can offer us the infrastructure we'll need to scale our concurrency to the thousands of simultaneous access, without affecting our database performance (with multiple queries for catalog searches, filters, updates, and mainly the checkout processes). Amazon's RDS, for example, can help us with scaling, replication and failover issues. Their ElasticCache can access frequently requested data, reducing database load. Microsoft Azure (Cosmos DB) offers low latency, geo-replication, high scalability, etc. Google cloud is not far behind in these services, with Cloud SQL, Cloud Spanner, Cloud CDN, etc.

4.1.3 Scalability: The level up to which we can grow depends on whether we do either of these:

1. **Vertical Scaling** (Scaling Up)
   - Adding more **hardware**: Upgrading our server with a CPU with more cores, add more RAM.
   - Using more efficient **software**: Better database normalization and relationships for faster queries. Optimizing the application code, to make it more efficient in resource utilization.
   - Improve **storage** speed: Investing in more and faster storage solutions like RAID arrays, SSDs, or NVMe drives for better I/O performance.

2. **Horizontal Scaling** (Scaling Out)
   - Adding **more servers**: Increasing the number of servers to distribute the load across multiple machines, basically creating a server cluster that works as a single system.
   - **Load Balancing**: Implementing load balancers to distribute requests evenly across multiple servers to prevent any single server from being overwhelmed.
   - **Database Replication**: Set up database replication to spread read queries across multiple servers, improving performance without burdening the main database server.
   - Using **cached layers**: Implementing caching solutions like Memcached or Redis to reduce database load by storing frequently accessed data in memory.

4.2 Usability Requirements

"Usability means making sure that something works well: that a person of average (or even below average) ability and experience can use the thing for its intended purpose without getting hopelessly frustrated." (Steve Krug)

4.2.1 Intuitive Interface: The user interface will be the easiest to navigate for users of all technical skill levels, with clear labels, instructions, and feedback mechanisms. It will be responsive, with seamless transitions between large or small desktop screens, as well as tablets or any number of cell phones, ensuring a comfortable experience for the user.

4.2.2 Consistent UX: The user experience (UX) will be intuitive, user-friendly and most importantly consistent across all parts of the system, with uniform styling, color swatches, functionality, etc. We don't want them to have the experience of going from one environment to another, as if jumping from country to country, culture to culture or driving on the left vs. on the right.

4.2.2 Responsive Design: The website will be built using responsive web design techniques to ensure that it displays correctly across devices with different screen sizes (desktops, tablets, smartphones). We will take the "Mobile-First" approach (prioritize mobile usability), but always providing a seamless experience across platforms. We're inclined to use the **Vaadim** framework, that focuses on user experience, with a rich set of UI components and server-side capabilities.

Later on in our evolution we will build a Mobile App. This will be both a native iOS and Android app, built using either **React Native**, Flutter, or Swift/Java/Kotlin, that will provide additional features like biometric login, QR code catalog browsing, and mobile payment options.

4.3 Security Requirements

"Security is a process, not a product." (Bruce Schneier)

4.3.1 User Data Protection

- **Encryption**: All sensitive user data and user account info (e.g., email, shipping address, passwords, payment information, etc.) will be encrypted.
- **Authentication**: We will implement a two-factor authentication (2FA), for admins and customers. The system will cache their username and password, but from time to time we will ask pertinent captcha questions. **Also, all user data will be sanitized and validated before processing.**

- **Session Management**: User sessions will have a brief and automatic expiration as a protection against session hijacking attacks.

4.3.2 Secure Payment Processing

- **PCI Compliance**: Our payment gateway will comply with PCI-DSS (Payment Card Industry Data Security Standard) for processing and storing credit card data.
- **Tokenization**: We will use tokenization to process payments securely, that way sensitive payment data will never be exposed during transactions.
- **Fraud Detection**: We will have fraud prevention mechanisms, such as CAPTCHA, IP address monitoring, and special red alerts for suspicious transactions.

4.4 Availability and Reliability

"Availability and reliability are two of the most important characteristics of a system. They are the keys to delivering a system that meets user expectations and withstands real-world conditions." (Martin L. Abbott & Michael T. Fisher)

4.4.1 Uptime: Our system should be up and running 99.9% of the time (24/7), minimizing downtime and ensuring constant availability for customers and administrators.

4.4.2 Redundancy: We'll have redundant servers, databases, etc., to ensure the system remains operational even in case of a hardware failure, as is often the case in the lightning capital of the world.

4.4.3 Backup and Recovery: We'll do daily and weekly backups to ensure data integrity, so that we can recover the system in case of disaster (hurricane, flood, lightning, etc.) and a disaster recovery plan.

4.4.4 Failover Mechanisms: In case of server failures, TCP-IP connection issues, etc., an automatic failover should occur to maintain continuous online service without delays.

4.4.5 Deployment: Initially we will host everything "on prem" (in-house server) but as need arises we will acquire space with a reliable host like GoDaddy or a less expensive one like Hostinger. As our number of sessions increases, (e.g. to avoid Denial of Service), we will scale up to the cloud, most likely AWS.

4.5 Compliance with Industry Standards (PCI, GDPR, etc.)

"Privacy regulations like GDPR aren't just about data protection; they're about the individual's control over their information." (Daniel J. Solove)

4.5.1 PCI Compliance: For secure processing of credit card payments, our system will comply with the Payment Card Industry (PCI) standards.

4.5.2 GDPR Compliance: We will also comply with the General Data Protection Regulation (GDPR), to ensure our users that they have control over their personal data, which they can delete, or modify.

4.5.3 Data Retention Policies: We will only store customer data as long as necessary and will apply automatic data deletion wherever applicable.

4.6 Maintainability and Support

"If it hurts, do it more often, and bring the pain forward. The more often you integrate, the less painful it will become." (Jez Humble)

4.6.1 Modular Design: Our microservices architecture will facilitate easy updates, bug fixes, and the addition of new features without affecting the overall system.

4.6.2 Documentation: This Requirements Document, as well as other manual pages will be provided to developers and sys admins, in regard to the system's architecture, database schema, and troubleshooting procedures.

4.6.3 Version Control: We will make extensive use of GitHub to track code changes, go back to previous versions, and keep consistent versions across different platforms (Win, Mac, Android, etc.)

4.6.4 Automated Testing: In chapter 8 we expand on testing a bit more. We'll do unit, integration, and acceptance tests to ensure the system is free of bugs, that will disrupt its functionality.

4.6.5 Support Plan: After deployment, we will be monitoring the system and provide updates, and security patches to keep the system secure and functional, as part of our maintenance plan.

4.6.6 Incident Management: In case of any problem, we have a plan for handling outages, security breaches, or other critical incidents, with timely responses and resolution.

4.6.7 Audit Trail: At GitHub we will maintain a history of changes to facilitate system support.

---

5. Technical Requirements

"Building reliable, scalable, and maintainable data systems is a complex task, but by carefully designing the architecture and considering the trade-offs, we can build systems that meet the needs of modern applications." (Martin Kleppmann)

Our goal is to ensure that our eCommerce system is robust, secure, scalable, and able to support any current and future of **Pikazon's** demands, as well as handling security and reliability, including responsiveness to future screen sizes.

5.1 The 3 Tier Architecture

"The three-tier architecture is a widely used pattern for designing enterprise applications. It separates the system into three layers: presentation, application, and data. This separation helps in organizing code and allows for more scalable and maintainable systems." (Ian Sommerville, Chapter 4, p. 96)

The context of this system is highly interactive, and **user centered** (personalized with ratings, wishlists, favorite items, forms of payment, etc.). It is also collaborative (reviews , recommendations), with dynamic interfaces (intuitive and seamless), rich media platforms, and most importantly a strong API integration connecting several services such as payment gateways, shipping providers, supplier, etc.

For now, we've rejected the latest Web 3.0 because it would move us away from a centralized server where we have control over our data, placing it instead of a "shared" distributed network, with blockchain technology (too advanced for us at the moment.)
For this reason, we've selected a 3-tier architecture. With a separation of concerns, we can increase scalability, security, and proficiency in our operations, such as data management, data analytics, 3rd party management, etc. As we saw earlier, data security is a big concern to us, and as we will see later, so is data storage.

5.1.1 Presentation Layer: Our front-end system includes the components that are responsible for interacting with the user, displaying data, and sending requests to the server. We will use technologies like HTML, CSS, JavaScript (we prefer Spring Boot over Node.js, React, Angular or Vue). For now, we will focus on Web Apps, but later on, when we expand to phone apps, where we will switch to React Native. Our code is viewable in all modern browsers (Chrome, Firefox, Opera, Edge, Brave, Waterfox, etc.) in any Operating System as well as mobile platforms (iOS, Android).

5.1.2 Application Layer: Our back end will handle the business logic, i.e., processing user input, performing business operations such as order management, product searches, as well as account management. We will disregard JSP and Node.js, in favor of using the Spring Boot framework because it's more scalable, robust, and built for enterprise-level Java applications.

5.1.3 Database Layer: Connected through the Application layer, at this level we handle all data-related operations such as storing, retrieving, updating, and deleting data (e.g., user profiles, product catalog, order history, etc., etc.). We will use MySQL for such queries.

5.2 Database Requirements

"A well-designed database ensures data consistency, integrity, and can efficiently handle high volumes of transactions." (Elmasri & Navathe).

5.2.1 Data Models: Our collection of data (related to products, customers, orders, etc.) will be organized through our Entity-Relationship Diagram. This ERD will define the tables in our DB and how they relate to each other (1 to many, etc.), e.g. one customer can place many orders, one category can include many books**.** Each entity will have a unique primary key (e.g., bookID, orderID), and none, one or many foreign keys, set to establish a relationship between the tables (e.g., customerID in the Orders table linking to the Customer table). A key reason to have primary keys is to optimize search performance. Also, to reduce redundancy we've normalized (3NF) our schemas, ensuring data integrity.

5.2.2 Book Inventory Database: The purpose of this database is to store the complete catalog of books, including their metadata (e.g., title, author, genre) and inventory data (e.g., stock levels, price). This database also tracks all customer orders, payments, and shipment statuses, it stores key order details and customer data and the book inventory. The following schema is a detailed description of each table in the DB that I created in MySQL Workbench (after which I present the ERD for said tables):

We start by creating the database, and the setting it up for usage:

```
create database pikazon;
use pikazon;
```

Now we create tables, with specific attributes. We also defined their type and size:

```
1. create table User(
```

```
    userID              int primary key not null auto_increment,
    userName        varchar(50),
    passwrd         varchar(50),        -- this will be confirmed and hashed, before storing it
    email           varchar(100),
    userRole        enum('customer', 'admin', 'staff'),
    firstName       varchar(50),        -- separate from userName (which belongs to login)
    lastName        varchar(50),
    phone           varchar(15),
    address         varchar(255),
    dateCreated     timestamp);

2. create table Book(
    bookID              int primary key not null auto_increment,
    title               varchar(255),
    descript            text,
    author              varchar(255),
    isbn                varchar(13),
    publisher           varchar(100),
    publicationYear     year,
    price               decimal(10,2),
    stockQtty           int,
    coverImage          varchar(255),
    categoryID          int,  foreign key (categoryID) references Category(categoryID));

3. create table Category(
    categoryID    int primary key not null auto_increment,
    categoryName  varchar(50) );

4. create table Orders(
    orderID             int primary key not null auto_increment,
    orderDate           date,
    orderStatus         enum('pending', 'shipped', 'delivered', 'canceled', 'returned'),
    totalAmount         decimal(10,2),            -- how much was spent
    shippingAddress     varchar(255),
    trackingNumber      varchar(50),        -- tracking number for the shipment
    paymentMethod       enum('debit_card', 'credit_card', 'paypal', 'cash'),  -- payment method
    paymentStatus       enum('paid', 'unpaid', 'disputed', 'refunded'),
    userID              int,  foreign key (userID) references User(userID));

5. create table OrderDetail(  -- the items in each order
    orderDetailID       int primary key not null auto_increment,
    quantity            int,                -- number of units of the book ordered
    pricePerUnit        decimal(10,2),      -- price of each unit at the time of the order
    bookID              int, foreign key (bookID) references Book(bookID),
    orderID             int,  foreign key (orderID) references Orders(orderID));

6. create table ShoppingCart( -- the current cart items for customers before they finalize an
order
    cartID              int primary key not null auto_increment,
    quantity            int,                -- quantity of the book added to the cart
    bookID              int,  foreign key (bookID) references Book(bookID),
    userID              int,  foreign key (userID) references User(userID));

7. create table Payments(
    paymentID           int primary key not null auto_increment,
    amountPaid          decimal(10,2),            -- amount paid by the customer
    paymentMethod       enum('debit_card', 'credit_card', 'paypal', 'cash'),  -- payment method
    transactionID       varchar(100),-- transaction ID from the payment gateway
    orderID             int,  foreign key (orderID) references Orders(orderID));

8. create table Reviews(         -- the customer reviews for books
    reviewID            int primary key not null auto_increment,
    Rating              tinyint,            -- given by the customer (from 1 to 5)
    Review              text,               -- the customer's written review
    dateWritten         timestamp,          -- date and time the review was submitted
    bookID              int, foreign key (bookID) references Book(bookID),
    userID              int,  foreign key (userID) references User(userID));
```

```
9. create table Promotions(             -- promotional codes and discounts
      promotionID          int primary key not null auto_increment,
      promoCode            varchar(20),  -- promotional code (e.g. "SAVE_20%", "BOGO", etc.)
      percentDiscount      decimal(10,2),      -- discount percentage applied to an order
      validFrom            date,               -- Start date for the promotion
      validUntil           date,               -- expiry date for the promotion
      minimumOrder         decimal(10,2) );    -- Minimum order amount to qualify for a promo

10. create table Shipping(       -- tracks shipping and delivery status for each order
      shippingID           int primary key not null auto_increment,
      shippingMethod       enum('standard', 'express'),
      carrier              enum('store_pickup', 'usps', 'ups', 'fedex'),
      trackingNumber       varchar(50),  -- tracking number for the shipment
      shippedSate          date,               -- date when the order was shipped
      expectedDeliveryDate date,        -- expected delivery date
      actualDeliveryDate   date,        -- actual delivery date
      orderID              int,  foreign key (orderID) references Orders(orderID));

11. create table WishList(       -- the books saved for future purchase
      wishlistID           int primary key not null auto_increment,
      dateCreated          timestamp,          -- when the book was added to the wishlist
      bookID               int, foreign key (bookID) references Book(bookID),
      userID               int,  foreign key (userID) references User(userID));

12. create table CustomerSupport(-- tracks customer support tickets and requests
      ticketID             int primary key not null auto_increment,
      problem              text);        -- problem with the book, the delivery, etc.
```

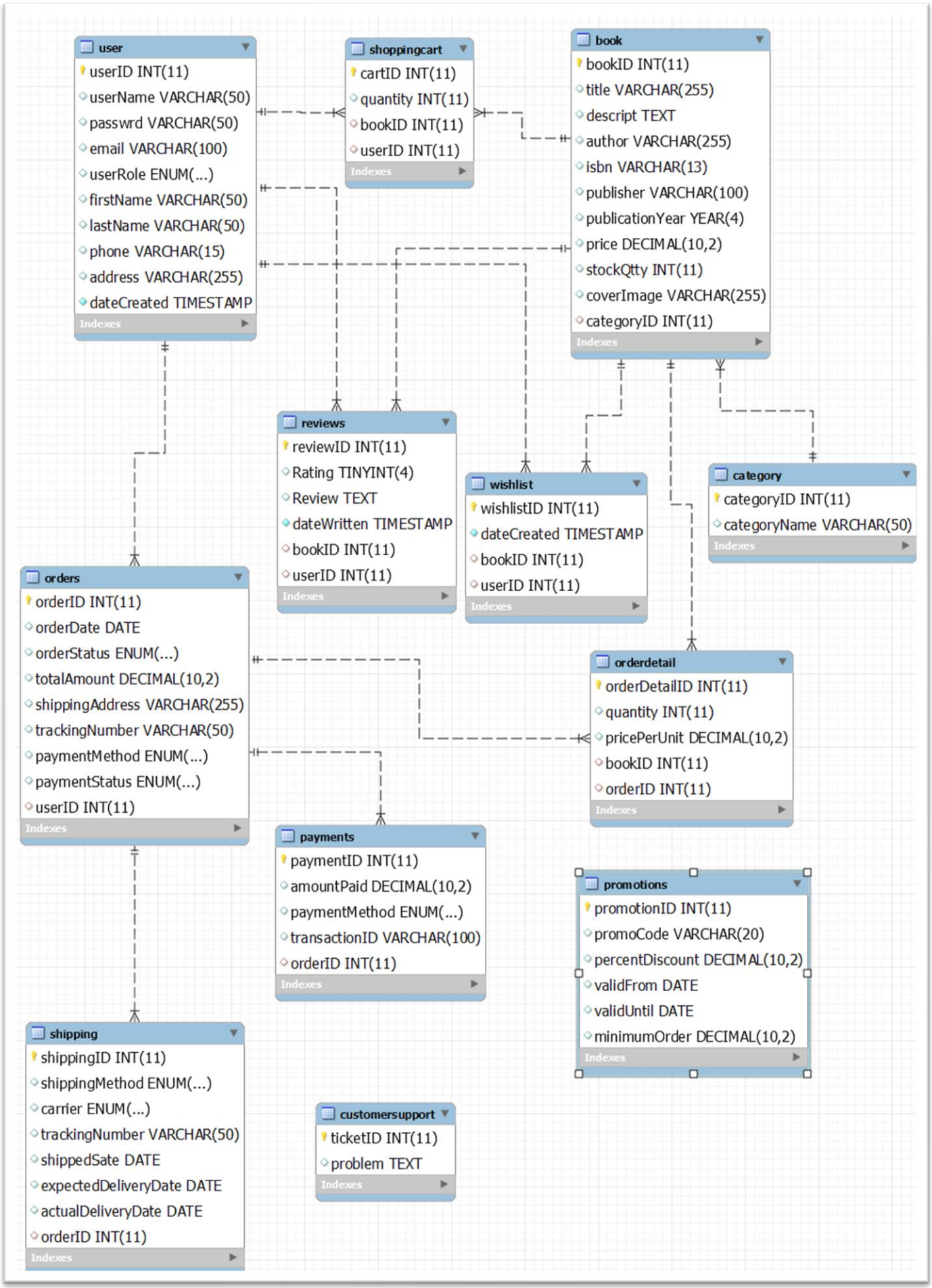
I probably need to create lots more tables, and relationships, but we're on time constraint here. There are many samples available in books like the following:

- "Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design" by M. J. Hernandez,
- "Fundamentals of Database Systems" by Ramez Elmasri and Shamkant B. Navathe
- "Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems" by Martin Kleppmann

Note: The above tables are my own concoction, similar to what I've seen in other ecommerce sites (e.g. Northwind Traders, for Microsoft Access) and schemas viewable at the following sites:

1. https://mavink.com/explore/Database-Schema-for-School-Management-System
2. https://fordfiringorder.com/
3. https://database.guide/what-is-a-database-schema/sakila_full_database_schema_diagram/
4. https://creately.com/diagram/example/huvdo5i22/opencart-database-schema-classic

This is the MySQL's Workbench reverse engineered ERD. (Subject to change).


user
userID INT(11)
userName VARCHAR(50)
passwrd VARCHAR(50)
email VARCHAR(100)
userRole ENUM(...)
firstName VARCHAR(50)
lastName VARCHAR(50)
phone VARCHAR(15)
address VARCHAR(255)
dateCreated TIMESTAMP
Indexes
shoppingcart
cartID INT(11)
quantity INT(11)
bookID INT(11)
userID INT(11)
Indexes
book
bookID INT(11)
title VARCHAR(255)
descript TEXT
author VARCHAR(255)
isbn VARCHAR(13)
publisher VARCHAR(100)
publicationYear YEAR(4)
price DECIMAL(10,2)
stockQtty INT(11)
coverImage VARCHAR(255)
categoryID INT(11)
Indexes
reviews
reviewID INT(11)
Rating TINYINT(4)
Review TEXT
dateWritten TIMESTAMP
bookID INT(11)
userID INT(11)
Indexes
wishlist
wishlistID INT(11)
dateCreated TIMESTAMP
bookID INT(11)
userID INT(11)
Indexes
category
categoryID INT(11)
categoryName VARCHAR(50)
Indexes
orders
orderID INT(11)
orderDate DATE
orderStatus ENUM(...)
totalAmount DECIMAL(10,2)
shippingAddress VARCHAR(255)
trackingNumber VARCHAR(50)
paymentMethod ENUM(...)
paymentStatus ENUM(...)
userID INT(11)
Indexes
orderdetail
orderDetailID INT(11)
quantity INT(11)
pricePerUnit DECIMAL(10,2)
bookID INT(11)
orderID INT(11)
Indexes
payments
paymentID INT(11)
amountPaid DECIMAL(10,2)
paymentMethod ENUM(...)
transactionID VARCHAR(100)
orderID INT(11)
Indexes
promotions
promotionID INT(11)
promoCode VARCHAR(20)
percentDiscount DECIMAL(10,2)
validFrom DATE
validUntil DATE
minimumOrder DECIMAL(10,2)
Indexes
shipping
shippingID INT(11)
shippingMethod ENUM(...)
carrier ENUM(...)
trackingNumber VARCHAR(50)
shippedSate DATE
expectedDeliveryDate DATE
actualDeliveryDate DATE
orderID INT(11)
Indexes
customersupport
ticketID INT(11)
problem TEXT
Indexes

5.3 API Integration

"APIs should be the primary point of interaction with the system, facilitating separation of concerns and service autonomy." (Sam Newman)

5.3.1 Internal APIs: Our RESTful APIs will allow the frontend to communicate with the backend services. Key APIs will include:

1. Authentication (for login, registration, session management)
2. Catalog (for searching, filtering, and viewing books)
3. Order (for placing and tracking orders)
4. Payment (for processing payments and refunds)

5.3.2 External APIs: This is how the system will integrate with third-party services. Key APIs will include:

1. Payment Gateway: We have to connect with payment services like Stripe, PayPal, or Square to process transactions securely.
2. Shipping: We will also integrate with logistics providers (e.g., FedEx, UPS, or the USPS) to calculate shipping rates and track deliveries.
3. Notifications: We might integrate with email/SMS services (e.g., SendGrid, Twilio) to send notifications for order confirmations, shipping updates, etc. Not decided yet.

5.4 Payment Gateway Integration

5.4.1 Supported Gateways: Our system connects to major payment gateways like Stripe, PayPal, or Square for processing credit card and alternative payments (e.g., digital wallets, bank transfers).

5.4.2 Tokenization: Our system uses tokenization to replace sensitive data with unique tokens that are useless outside a specific transaction.

5.4.3 PCI-DSS Compliance: Our system will comply with Payment Card Industry Data Security Standard (PCI-DSS) to ensure secure handling of credit card information.

5.5 Backup and Recovery

"Continuous delivery is about ensuring that the system remains in a deployable state throughout its lifecycle, no matter the changes being made." (Jez Humble)

5.5.1 Automated Backups: Our system implements automatic, daily and weekly backups of all critical data, including the database, user profiles, orders, and transaction logs.

5.5.2 Redundant Storage: Backups will be stored in a redundant, secure location. We will use cloud storage in the future (e.g., AWS S3, Azure Blob Storage) that keeps data over several data centers.

5.5.3 Disaster Recovery Plan (DRP): Our disaster recovery plan outlines procedures for restoring services in the event of data loss, server failure, or cyberattacks. This plan includes:

- Recovery Time Objective (RTO): How quickly can we come back after a disaster (target RTO: < 1 hour).
- Recovery Point Objective (RPO): The tolerable amount of data loss (target RPO: < 5 minutes).

6. User Interface Requirements

“The user interface design should be based on a clear understanding of the user’s tasks and the system’s operational context. Effective interface design can significantly improve user productivity and satisfaction.” (Ian Sommerville, . Chapter 16, p. 448)

6.1 Website Interface

"Users spend most of their time on other sites. This means that they will be the most comfortable with designs that follow established conventions." (Jakob Nielsen)

"It’s not that users don’t read; they just don’t read. They scan. They glance. They pick out keywords." (Steve Krug)

Our website is the primary way online customers interact with the bookstore. Our design prioritizes simplicity and aesthetics to encourage users to explore the catalog, make purchases, and manage their accounts.

6.1.1 Design Principles: We offer a clean, modern design with a focus on a minimalist interface, featuring prominent navigation bars for categories, search functionality, and a shopping cart.
6.1.2 Navigation: The website has an intuitive menu with clear categories (e.g., fiction, non-fiction, bestsellers), a powerful search bar, and filtering options (e.g., genre, author, price).
6.1.3 Book Detail Pages: Each book has its own page with detailed information, including a cover image, a synopsis, the author’s bio, reader’s reviews, and if it’s in stock or backordered.
6.1.4 Customer Account Area: Users’ account has a dashboard to track orders, update payment methods, manage their wishlist, and view their purchase history.
6.1.5 Responsiveness: Our site is fully responsive, adapting to various screen sizes and orientations.

6.2 Web App for Cell Phones

Our web app is a mobile-optimized website that looks and feels like an app when accessed through a phone’s browser. It does not install on the phone but runs on any platform (like a progressive web app). In the future we will incorporate a native app experience, that will allow customers to use “biometrics” to log in to their account.

6.2.1 Search and Browse: Easy access to the catalog with swipe gestures, and keyword search.
6.2.2 Mobile Payment Integration: The checkout process allows for payment with Apple Pay & Google Pay.
6.2.2 QR Code Scanning: Our in-store customers can use our app to scan QR codes in books for more details, availability, reviews, or to add them directly to their cart.

6.3 In-store Interface for Employees

"The goal of interaction design is to create products that are useful and satisfying to the user." (Alan Cooper)

The in-store tablet system will help employees handle transactions, manage inventory, and help customers.

6.3.1 Employee Login: Employees must log in with secure credentials to access features based on their role (e.g., cashier, manager) for security and auditing purposes.
6.3.2 POS System Integration: Our tablets offer a touchscreen interface for the Point-of-Sale (POS) system, allowing our staff to process transactions, apply discounts, or check inventory in real time.

6.3.3 Inventory Management: A back-end system that enables our staff to update stock levels, accept new shipments, and check the availability of books.

6.3.4 Customer Assistance Interface: They will be able to search the catalog, provide suggestions, etc.

6.4 Accessibility Standards

"The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect." (Tim Berners-Lee)

Our system interface (for web & mobile) complies with accessibility standards for customers with disabilities**.**

1. There are text alternatives for non-text content (e.g., alt text for images).
2. There is a good contrast ratio between text and background colors.
3. There is screen reader support.
4. Finally, customers can adjust the font size across all interfaces for better readability.

---

7. Data Requirements

7.1 Data Storage and Retention Policies

7.1.1 Data Storage: As an RDBMS, MySQL is robust and scalable enough to store all functional data, including inventory, customer information, orders, and transactions. But in the future, for scalability and redundancy, we will consider using cloud storage solutions (e.g., AWS, Google Cloud, Azure) to handle larger volumes of data with high availability.

7.1.2 Data Retention Policies: The Retention Periods:

1. Customer Data will be retained for up to 7 years, to comply with tax and accounting regulations.
2. Order History: We'll keep this data for 5 years for customer reference and support purposes.
3. Inventory Data: This we'll keep indefinitely to track historical inventory levels and trends.

7.1.3 Backup Procedures: It's said Florida is the lightning capital of the world, and for that reason there's a high probability of having a power surge (or power failure) that can disrupt our operation, and most importantly cause data loss. So, we've scheduled regular database backups, on a daily or weekly basis. We will store these backups in a secure location at the store (daily), or at a different physical location or cloud storage, for weekly backups. We will also test these backups, and the consequent recovery procedures to ensure data can be restored accurately and promptly when needed.

7.2 Data Import and Export Procedures

7.2.1 Data Import Procedures: In the rare occasion for bulk uploads of inventory, customer information, and order data into our database, we can accept various data formats such as CSV, XML, and JSON. And of course, we will ensure data integrity and prevent errors. We will check for duplicate records, invalid data formats, and missing required fields.

7.2.2 Data Export Procedures: In case we need to export data, we will use commonly used formats, such as CSV, Excel, or JSON, (XML is a little out of date). This data will be securely transmitted (to

facilitate data analysis and reporting, e.g. sales summaries, inventory levels, or customer activity), and it will also be protected from unauthorized viewers.

7.2.3 Integration with External Systems: We will implement a seamless data exchange between the bookstore's system and external systems, such as publishers, payment gateways, accounting software, or third-party vendors.

7.3 Data Synchronization (In-store vs. Online)

7.3.1 Synchronization Mechanisms: There will be real-time data synchronization between in-store systems (POS) and online platforms because we will be using the same database. This will ensure that inventory levels, pricing, and order statuses are instantly updated.

7.3.2 Conflict Resolution:
A customer might go by Tony Rojas sometimes, or Antonio Rojas at other times, but it's the same customer. There's no need to maintain 2 separate files.

1. **Data Conflicts**: We will develop methods to solve such problems, as well as data discrepancies between in-store and online systems, by setting priorities on which data source weighs in more.
2. **Audit Trails**: An audit is the best way to track discrepancies in our client's data and identity. It solves synchronization issues.

7.3.3 Performance and Scalability:

1. **Scalability**: We designed our synchronization processes to scale up, with the growth of the bookstore's data volume and transaction frequency.
2. **Performance Monitoring**: We will monitor the performance of data synchronization processes to ensure they do not impact system performance or user experience negatively.

---

8. Testing and Validation

"Testing is an infinite process of comparing the invisible to the ambiguous, so as to avoid the unthinkable happening to the anonymous." (Cem Kaner)

Adhering to formal testing standards, based on the guidelines from the IEEE 29119, ISO, or ISTQB (International Software Testing Qualifications Board) for test case development, our software will be tested on functional and non-functional requirements, to ensure it meets their standards.

For **Pikazon**, a small local bookstore, the primary source for **Functional** requirements will be the Owner or Manager who has the most direct understanding of the business's goals, customer needs, inventory management, and online sales strategies, and can compare the system's intended and expected behavior versus the actual one.

The primary sources for **Non-Functional** requirements are the manager (on business operations and sales), the customers (having their needs and expectations met, as well as their security concerns), the employees (being able to perform front/back-office tasks) and the publishers & vendors (in regard to inventory integration, order

fulfillment, or digital catalogs). These stakeholders can best describe performance thresholds, security standards, scalability needs, and other quality attributes of the system.

8.1 Test Cases

Here we will focus on specific types of tests and the acceptance criteria used to validate our software.

8.1.1 Test ID: A unique identifier for the test case
8.1.2 Description: A brief overview of what the test case will do.
8.1.3 Steps: How are we going to conduct the test, step by step.
8.1.4 Expected: The expected outcome for each step or the entire test.
8.1.5 Actual: The actual outcome after the test is run.
8.1.6 Pass/Fail: Whether the validation test passed or failed.

8.2 Test Scenarios

Test scenarios, on the other hand, are broader (higher level) than individual tests. Common scenarios include:

8.2.1 Functional Testing: The system has to function according to the requirements.
8.2.2 Edge Case Testing: Testing unusual or extreme inputs (e.g. DoS) to ensure robustness.
8.2.2 Security Testing: Testing for vulnerabilities (e.g. when hackers execute arbitrary code on the server by sending a malicious HTTP request containing specially crafted commands to the web application) and security risks (e.g. In 2013 Target's network was compromised by the credentials of a vendor in charge of the HVAC system. Russian hackers gained access Target's network and then to the credit/debit card information of over 40 million customers.)

8.3 Acceptance Criteria

The conditions that our program has to meet to be considered acceptable by the stakeholders. They include:

8.3.1 Functionality Validation: Verifying that each feature works as expected.
8.3.2 Non-Functional Requirements Validation: Checking performance, security, usability, etc.
8.3.3 Compliance and Regulatory Standards: Ensuring the software adheres to industry standards.

8.4 User Acceptance Testing (UAT)

Here is where the beta testers test the program to make sure that it meets their needs and expectations, before its final deployment. The main criteria are:

8.4.1 Environment: these tests are conducted in a "as-close-to-real-life" environment as possible
8.4.2 Testers: People (staff) representing the target audience (customers, staff & managers).
8.4.3 Cases: Real-world cases that system users are likely to experience
8.4.4 Sign-off: is the formal acknowledgment that our system is ready for (pre) deployment.

8.5 Performance Testing

Performance testing evaluates the software's stability (response time, transaction rates), scalability, and resource usage (memory and CPU utilization) under various conditions. Key types of performance metrics include:

8.5.1 Load: Making sure the system can handle higher user loads without crashing.
8.5.2 Stress: Finding the breaking point under extreme loads
8.5.3 Scalability: See if the system scales up well, as the workload increases

8.5.4 Latency and Throughput: how quickly can our system respond to user requests and how many transactions it can handle in a set time.

9. Appendix

- **Minimal Hardware Configuration**:
  - CPU: 4-core processor.
  - RAM: 16 GB.
  - Storage: 2x10 TB WD external HD, (one is for constant daily backups).
  - Network: 1-5 Gbps internet connection. Charter Communications.
- **Optimal Hardware Configuration**:
  - CPU: 16-core processor.
  - RAM: 32-64 GB.
  - Storage: 5x10 TB WD RAID 5, as a NAS (with automatic backups).
  - Network: 5-100 Gbps internet connection with fiber optic.

Quotes' Citations (in order of appearance)

- Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson.
- Isaacson, W. (2011). Steve Jobs. Simon & Schuster.
- Drucker, Peter F. *Management: Tasks, Responsibilities, Practices*. Harper & Row, 1973.
- Shneiderman, B., & Plaisant, C. (2004). Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley.
- Bezos, Jeff. *The Everything Store: Jeff Bezos and the Age of Amazon*. by Brad Stone. Little, Brown and Company, 2013.
- Kleppmann, M. (2017). *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media.
- Krug, S. (2014). *Don't make me think, revisited: A common sense approach to web usability* (3rd ed.). New Riders.
- Schneier, B. (2000). *Secrets and lies: Digital security in a networked world*. Wiley.
- Abbott, Martin L., and Michael T. Fisher. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, 2016.
- Solove, D. J. (2008). *Understanding privacy*. Harvard University Press.

- Humble, J., & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Addison-Wesley.
- Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson.
- Newman, Sam. (2021). *Building microservices: Designing fine-grained systems* (2nd ed.). O'Reilly Media.
- Nielsen, J. (2000). Designing Web Usability: The Practice of Simplicity. New Riders.
- Cooper, A. (1999). The Inmates Are Running the Asylum. Sams Publishing.
- Berners-Lee, T. (2004). The Design of Web Pages and Applications. Available from W3C.
- Kaner, C., Falk, J., & Nguyen, H. Q. (1993). Testing computer software (2nd ed.). Wiley.