CS7314 Homework #1

Name: Bingying Liang
ID: 48999397

Jan 29 2023

Part 1: Problems from our textbook

1. Problems 1.3: Use the checklist in Section 1.4 and your personal goals to see if you need to review any background knowledge. If so, construct your individual study plan to get yourself ready for the rest of the book.

   Solution: Due to the checklist in Section 1.4, I want to improve my computer science and software engineering knowledge, I need to study Finite-state machines, Execution flow and data dependencies, Some formalisms, pattern matching, learning algorithms, and neural networks. For software engineering, I need to study some knowledge about the release, support, waterfall, spiral, incremental, iterative, extreme programming (XP), General awareness of software management and system engineering issues, including economic consequences of project decisions, tradeoffs between different objectives and concerns, feedback and improvement mechanisms, optimization, etc.

   My study plan:

   Schedule: Allocate specific times for each topic based on your familiarity and the complexity of the subject.

   Resources: Identify textbooks, online courses, tutorials, and practical exercises for each topic.

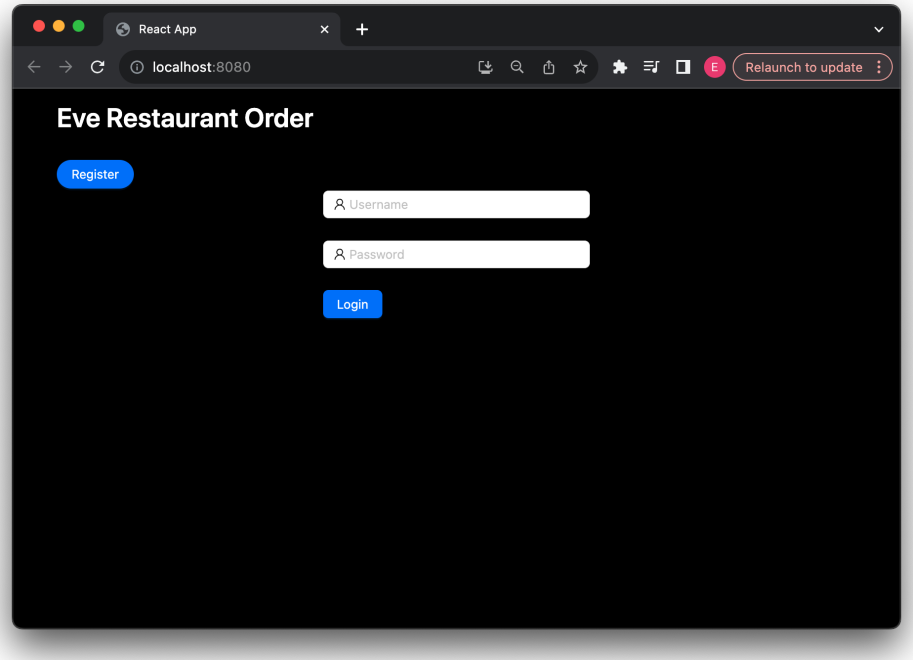
   Progress Tracking: Set milestones to track your progress in each area.

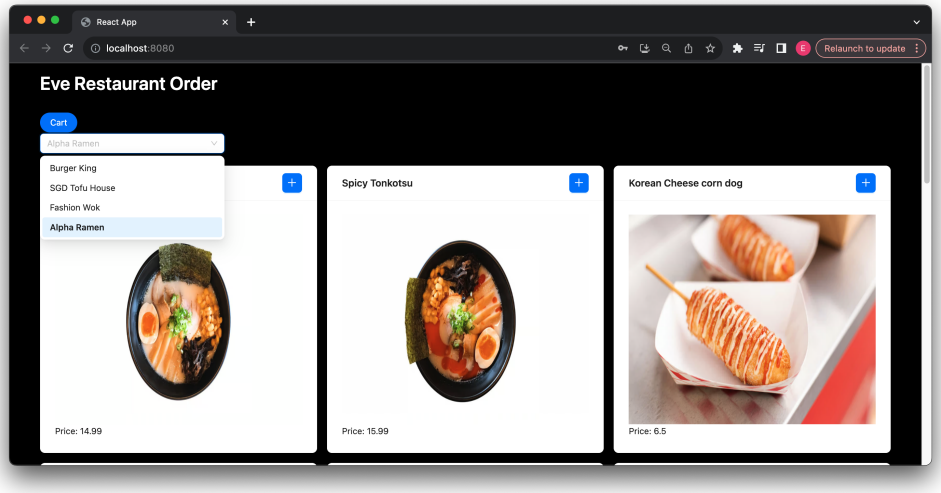
   Practical Application: Try to implement small projects or exercises related to each topic to enhance understanding.

2. Find a piece of software (e.g., something you wrote for another class, for work, for hobby, etc.), perform some (informal/ad hoc) testing and inspection (that would require access to the source code, or at least the description/specification/documentation of the software). Then briefly describe your activities and findings, paying attention to similarities and differences between inspection and testing.

   If you have some difficulty finding such a piece of software, try some open source ones via Web search. Alternatively, you can test/inspect a Web application/Website or Web pages.

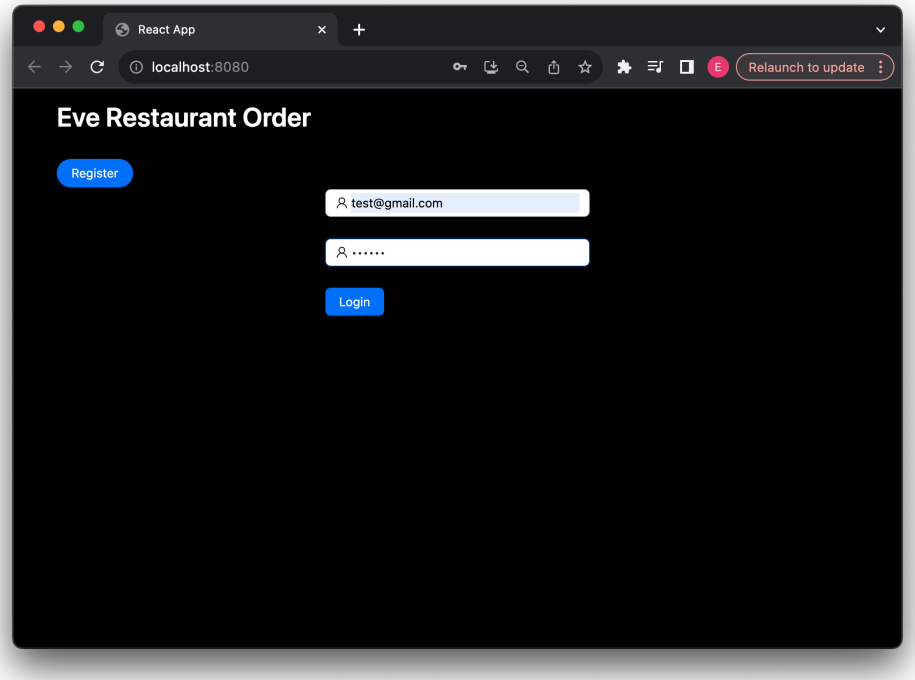
Solution: (a) Finding a Piece of Software I chose my project an online ordering system, which includes functionalities like menu browsing, selecting food items, placing orders, and making payments.

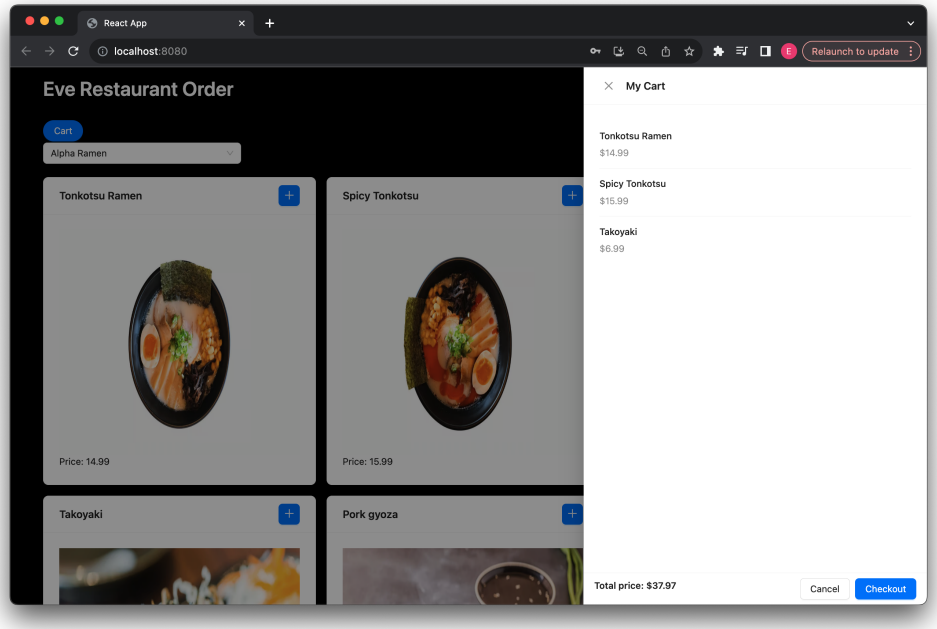




(b) Performing Informal/Ad Hoc Testing

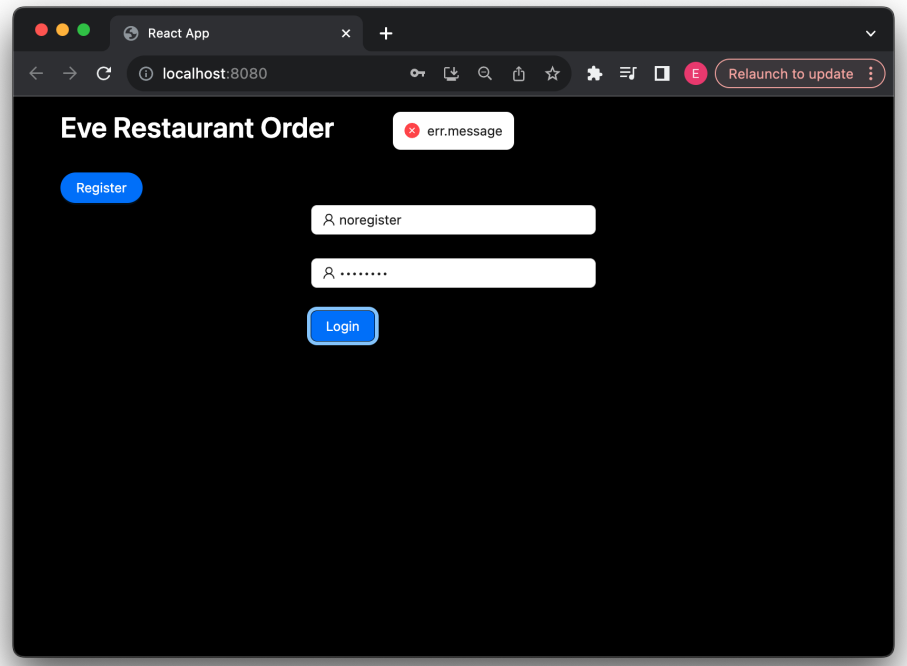
i. User Perspective Testing: Try using the system as a customer would. Browse the menu, add different dishes to the cart, and try payment. Observe if the system processes orders correctly and if there are any error messages or crashes.





The Informal testing shows works fine.

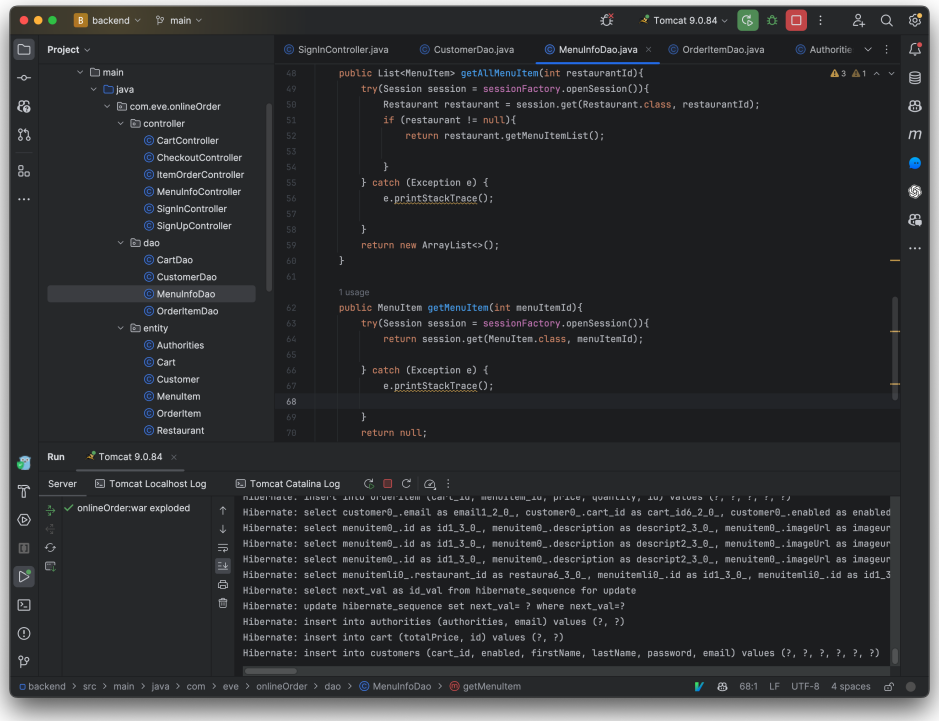
ii. Exceptional Input Testing: Input abnormal or non-standard data, such as invalid account information, to check if the system handles these scenarios properly.



The invalid account cannot login in. The system can handle these scenarios properly.

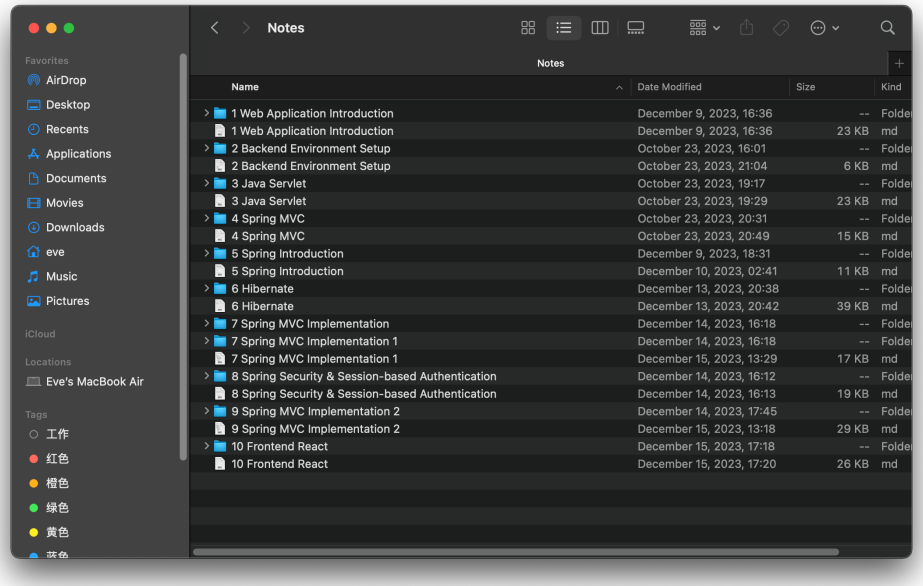
(c) Performing Inspection

i. Source Code Inspection: Review the system's source code for potential security issues (like insufficient input validation), check if the code is well-structured and commented on, and if it follows coding standards.



The comment is not clear or well-structured, and not very easy for a developer to read.

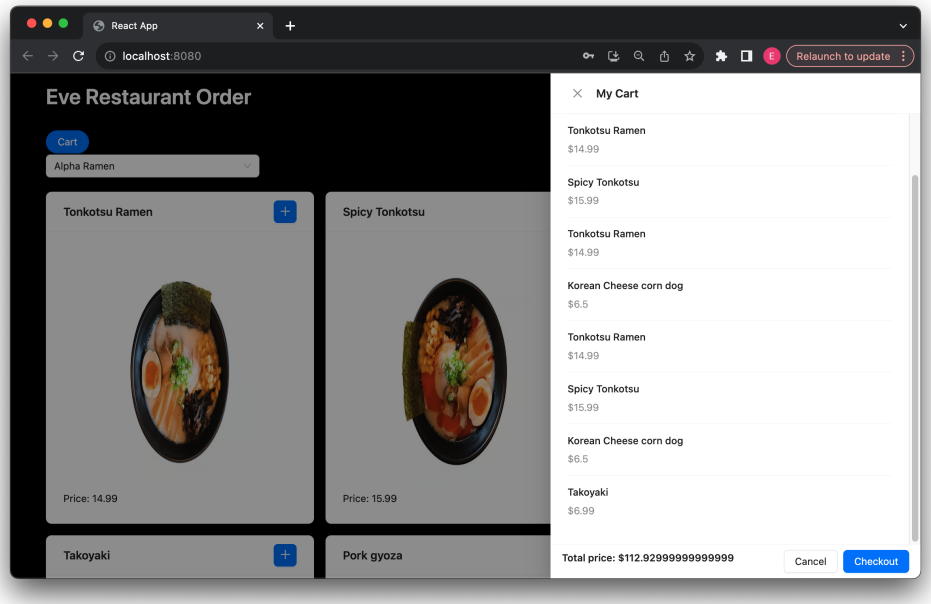
ii. Documentation and Specification Review: Ensure that the development documentation is clear and describes the system's functionalities and limitations in detail.



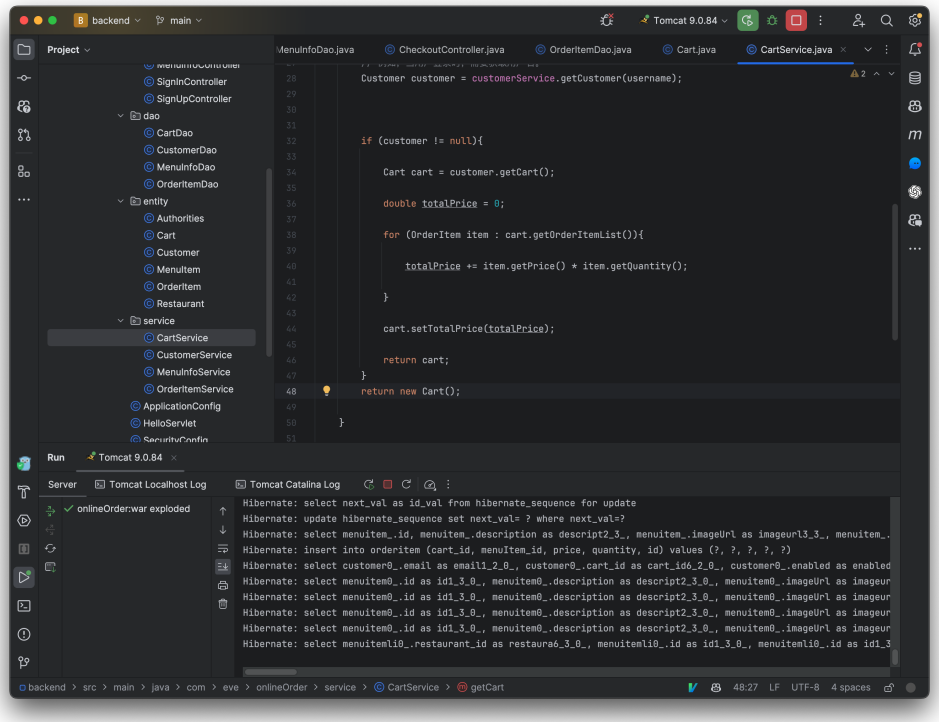
The project does not have professional documentation just some personal notes. The function of the system can easily be from code name, but still limitate the details.

(d) Describing Activities and Findings

i. Testing Findings: Issues find the checkout money about decimal point accurate too many places.



ii. Inspection Findings: There could be unhandled exceptions in the code.



The calculate of the money should add some limitations.

(e) Comparing Inspection and Testing

i. Similarities: Both are aimed at identifying and resolving issues within the software to enhance user experience and system stability.

ii. Differences: Testing primarily focuses on the software's operation and user interaction, capable of uncovering functional and non-functional problems; inspection, on the other hand, is concerned with the quality and structure of the code, as well as the accuracy and completeness of the documentation.

Part 2: Option B

If you do not have too much to report in the "practice" part, or if you don't feel comfortable discussing it, you can do one of the following:

- Perform a thorough individual study based on detailed documents describing QA/testing practices in a company or an organization, and write an essay similar to the above for this company (instead of your own).

- Select a chapter from Ch.13-16, or related papers or book chapters on similar topics, and write your own summary of the chapter/paper.

See the online list of related books/publications for additional reference.
(This is probably more appropriate for full-time students. Please make sure it is your own summary, based on your understanding of the subject after reading/studying, not the abstract, summary, or similar things directly from the original source you have just studied.)

You summary/essay should be around 2-3 double-spaced pages.

Solution: I chose Ch.13 and wrote my own summary of the chapter. The summary in the next new page.

Summary of Chapter 13: Defect Prevention and Process

Introduction: A Paradigm Shift in QA

Chapter 13 begins by contrasting the traditional reactive approach to Quality Assurance (QA) with a proactive stance, introducing the concept of defect prevention in software development. This paradigm shift is crucial for understanding the chapter's advocacy for early intervention in the QA process. It underscores the importance of recognizing and addressing potential issues before they become ingrained defects, thereby reducing the cost and effort associated with post-development fixes.

The Economic Impact of Late-Stage Defect Handling

The chapter dedicates significant attention to the economic implications of managing defects at later stages of software development. It presents a compelling argument backed by data and case studies showing that the costs associated with fixing defects escalate exponentially when they are not addressed early in the development lifecycle. It discusses the ripple effect of late defect discovery, leading to increased time-to-market, additional resource allocation for fixes, and potential reputational damage. The key takeaway is that an upfront investment in defect prevention can yield substantial long-term savings and efficiency gains.

Assessing the Limits of Traditional Testing

This section provides a critical assessment of the limitations inherent in traditional testing methods, particularly in the early development phases where executable code is not yet available. The chapter provides examples of scenarios where these limitations manifest, such as in requirement validation and design reviews. It also discusses alternative approaches that can complement traditional testing, like static analysis and model-based testing, which can be more effective in the early stages.

Comprehensive Strategies for Defect Prevention

The chapter then delves into two overarching strategies for defect prevention:

Error Blocking: This involves establishing robust processes and utilizing tools that can prevent the occurrence of errors that may lead to defects. The chapter discusses various methodologies and tools that have been effective in error blocking, such as rigorous code reviews, automated static analysis tools, and adherence to coding standards.

Source Removal: A deeper approach that aims to address the root causes of errors. This section expands on strategies such as refining requirements gathering processes, improving design methodologies, and enhancing team collaboration and communication. Real-world case studies are provided to illustrate how organizations have successfully implemented source removal strategies.

The Role and Execution of Root Cause Analysis

Root Cause Analysis (RCA) is extensively covered, with the chapter providing a detailed explanation of how to effectively conduct RCA. It breaks down the steps involved in identifying error sources, analyzing the underlying causes, and developing strategies to mitigate these errors. Challenges in effectively implementing RCA are discussed, along with strategies to overcome these obstacles, such as fostering a culture of continuous improvement and learning from past mistakes.

The Crucial Role of Education and Training

This section of the chapter underscores the importance of human elements in defect prevention. It explores how education and training can significantly reduce the likelihood of defects being introduced into software systems. The chapter suggests various approaches to training, including workshops, seminars, and hands-on sessions, which can be tailored to address specific areas of need within a software development team. It also highlights the role of continuous learning and upskilling in keeping pace with evolving technologies and methodologies.

Seamless Integration of Defect Prevention in Development Processes

The chapter concludes with a discussion on integrating defect prevention strategies into the fabric of the software development process. It advocates for the adoption of processes and methodologies that inherently reduce the likelihood of defects. This includes adopting agile practices, continuous integration, and test-driven development. The chapter also touches upon the importance of organizational culture in supporting these practices, emphasizing the need for leadership buy-in and a shift in mindset towards quality-first development.

Reflections and Practical Insights

In the final part of the chapter, there's a reflective analysis of the broader implications of adopting a defect prevention approach. The author shares insights from industry experts and case studies from successful implementations of defect prevention strategies. These anecdotes provide a real-world perspective on the challenges and rewards of shifting to a proactive QA approach.