
wikiHow to Study For Exams

CPEN 321

Recap + Midterm Prep

Where Are We?

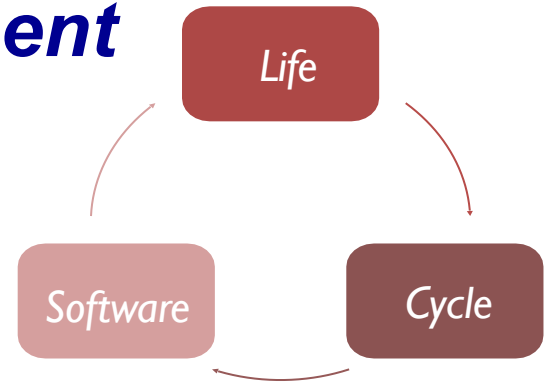
Design Thinking
- Processes
- Requirements
- Design

Coding
- Teamwork, version control
- Code reviews, code anti-patterns

Quality Assurance
- Testing
- Analysis

Deploy ment
- Continuous Integration, DevOps

Reminder: Popular Software Development Process Models



No planning

- **Code-and-fix**: write code, fix it when it breaks

Sequential

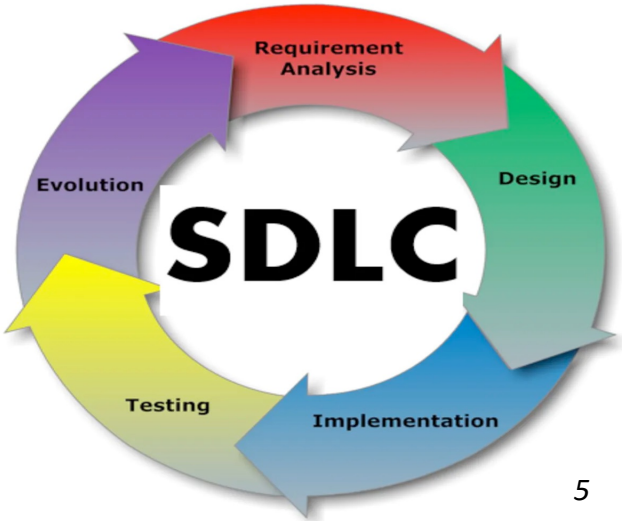
- **Waterfall:** perform each phase in order (~1970s)

Iterative

- **Staged Delivery:** waterfall-like beginnings, then, short release cycle
- **Evolutionary prototyping:** develop a skeleton system and evolve it for delivery
- **Spiral**: triage/figure out riskiest things first (1988)
- **Agile**: *a family of principles* promoting adaptive planning, evolutionary development, early delivery, and continuous improvement (1970-2005+)
  - Most popular: **Scrum** and **Kanban**

Processes: Main Message

- Customize the processes depending on the product, organizational culture, team structure, needs, etc.
  - Process models are often combined or tailored to the environment
  - Think how much time to spend on each task and in which order!
- Follow processes, but do not over-emphasize process over product
  - Don't become a "slave" of the process



Reminder: Defining Requirements in This Course

Functional

- Actors, functional requirement names (use cases), use case diagram
- Lightweight formal use case specification of each requirement:
  - name, short description, primary actors, success scenarios, failing scenarios
- Mockups, if helpful/needed

Non-Functional (at least 2)

- Specification for each requirement:
  - Textual description
  - Justification (why needed)
  - Validation approach (how to confirm)

Design Layers

High-level architecture

Detailed (program) design

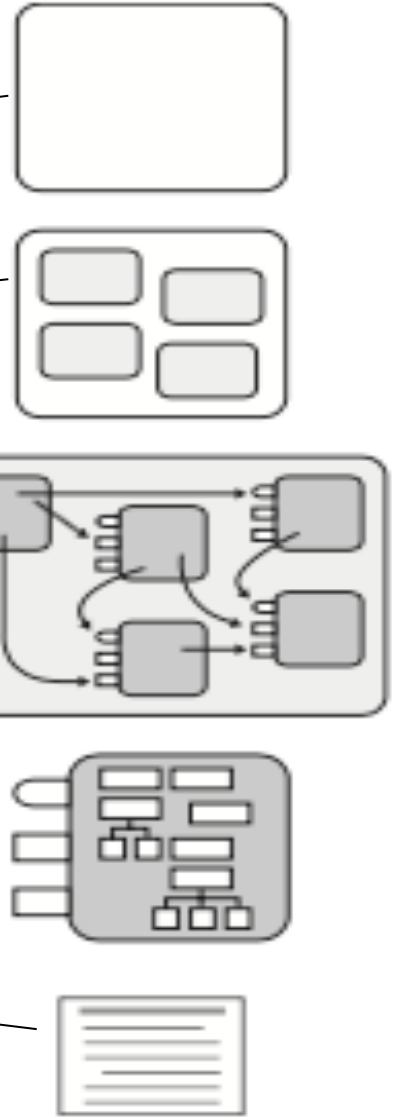
System as a whole

Division into modules/ packages

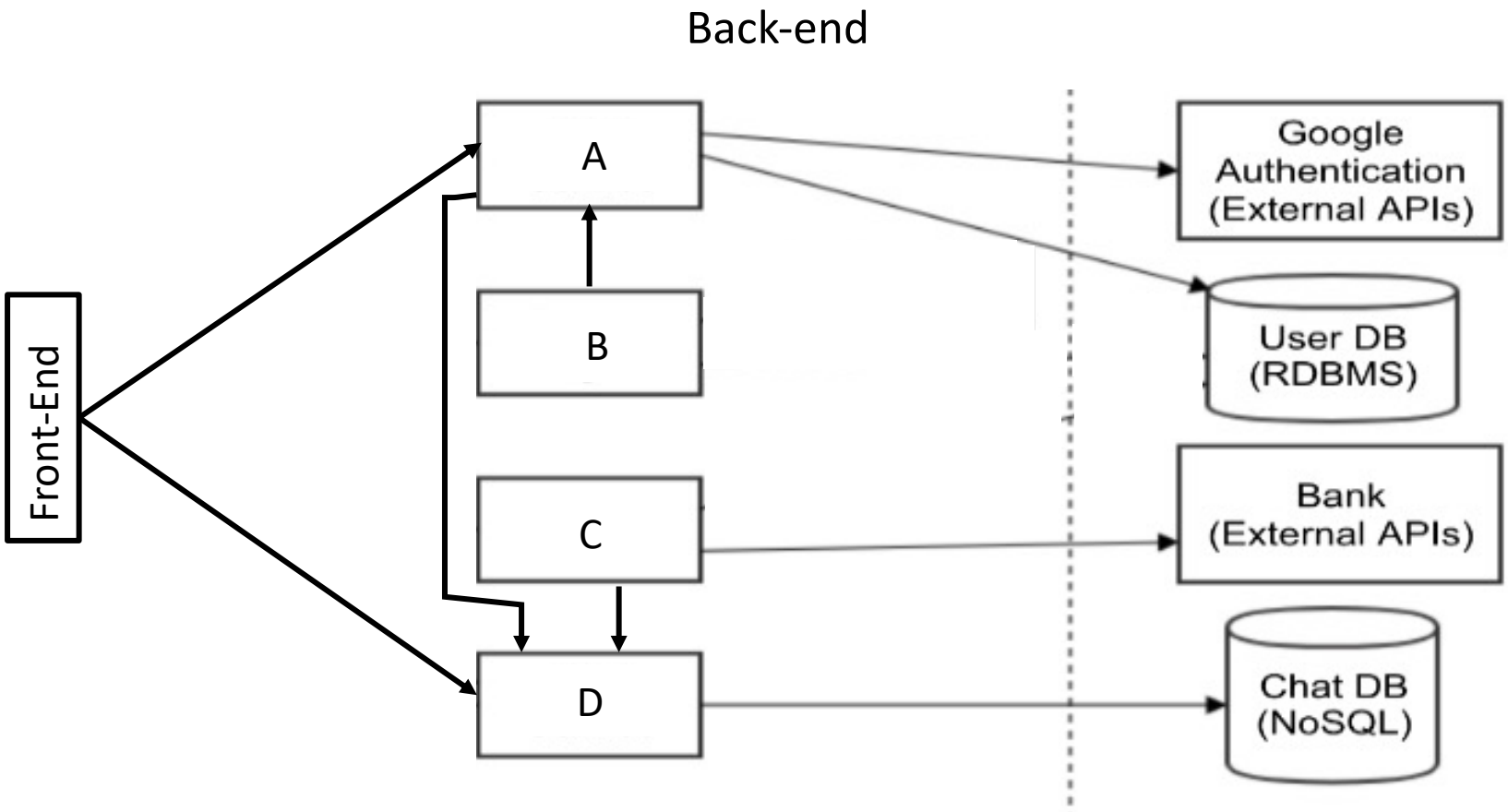
Classes within packages

Data and routines within classes

Internal routine design



Reminder: Design Diagram (Non-UML)



Generative AI

Assignment quote:

"ChatGPT is one of the greatest inventions of the current century. Humans need to find a way to use it in a good way instead of banning it."

What is in the Midterm?

- Processes
- Basic UML
  - use case, sequence, class diagrams
- Requirements
  - types of requirements
  - actors, use cases
  - use case diagrams
  - formal requirements specifications
- Design
  - components
  - Interfaces
  - design plots
  - sequence diagrams
  - REST, microservices

- ***Closed book***
- ***Allowed cheatsheet: A4/Letter format, 1 page, single side, hand-written only, has to be submitted***
- ***Student ID Required***

Best Way to Prepare

- Go over the lecture material and project milestones
  - **For the requirements and design**, take any project (e.g., of your peer-team) and define use-cases, non-functional requirements, main components, interfaces, and interactions between interfaces, as was done in the lecture notes and M2-M3 milestones.
  - Express them formally**,** as formal requirement specification or **UML** diagram (use case, class, and sequence), as appropriate
  - Read TA notes with most common mistakes
  - Discuss with team mates advantages and disadvantages of each solution
  - **For processes,** recall the main steps, ideas, and principles

Lab on Wednesday

- Will be optional (no attendance will be taken)
- TAs will be there to answer any questions
  - Treat it as office hours

Midterm Instructions (almost final)

RULES:

1. Please legibly write your name and student number on this page.
2. When you receive the signal to start, please legibly write your name and student number **on each page**.
3. There are a total of 100 marks and 4 questions.
4. Total time is **75 minutes**.
5. The quiz is closed-book; a one-page, A4 or Letter format, single-sided, hand-written cheatsheet is allowed. It has to be submitted with this booklet.
6. There are 16 pages in this booklet, including this page. Write all your answers in the booklet, either in the space provided for each question or on the blank pages provided at the end. If you use blank pages at the end, specify it in the first part of your answer and also write the number of the problem that you are continuing to solve next to your continued answer.
7. **Ensure that your handwriting and drawings are readable**. No points will be given if the handwriting and / or drawings are illegible.

Question	Marks	Achieved
1	20	
2	10	
3	30	
4	40	
TOTAL	**100**	

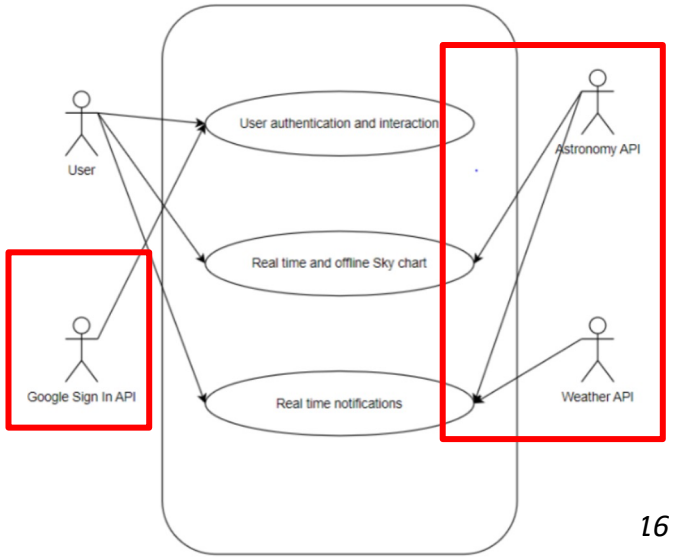
Questions so far?

Agenda for the Rest of Today

- Some common mistakes
- A design exercise
- M4 (MVP) spec

Use Cases and Actors

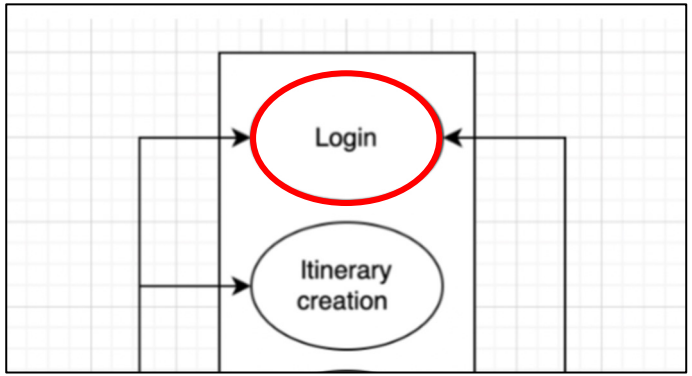
- Use cases are “active” (phrased as verbs not nouns)
  - E.g., "Study groups" => “Manage study groups”
- KISS: If the app does not need 3 actors, it is fine to have 1 or 2 (whatever is needed)
- Actors are users of the app, **not app developers** who interact with code, fix bugs, etc.
- Use case diagram does not have arrows between actors and use cases.
  - External APIs are not active users of the app (having arrows pointing to use cases is thus also counter-intuitive)



See lecture notes "W3 L2 More on RE" (slides 17-25) for more examples of mistakes in use case diagrams

Use Case Name == Functional Requirement Name

- Use case names do not match the names of functional requirements



Functional Requirements:

Name of Requirement: User Registration and Authentication
Short Description of Requirement: Users should be able to create accounts and log in securely.
Primary Actor(s): Travelers, Admin

Success Scenarios

- Scenarios should not be a high-level description of the cases but **a sequence of execution steps**

Success Scenario(s):

- Users create groups and successfully collaborate on trip planning.
- Group members can make joint decisions on itinerary items.

Failure Scenario(s):

- Users encounter errors while trying to create or manage groups.

How?

Success Scenario(s):

- User can search for events by category (e.g., music, sports, art).
- User can search for events by location (e.g., city or region).
- User can search for events by date range (e.g., this weekend, next month).
- User can search for events using keywords (e.g., "concert," "yoga class").

Success Scenario: Well-defined Sequence of Steps

Main Success Scenario:

1. Student selects “Register New Courses” from the menu.
2. System displays list of courses available for registering.
3. Student selects one or more courses he wants to register for.
4. Student clicks “Submit” button.
5. System registers student for the selected courses and displays a confirmation message.

See lecture notes "W3 L1 RE" (slides 46-47) and "W3 L2 More on RE" (slides 8-9) for examples of how to specify requirements correctly

Failure Scenario: Each Corresponds to a Certain Step in Success Scenario

- Each failure point corresponds to a failure in a particular success scenario steps
- Each failure point describes how the failure is handled (what the user sees)

Main Success Scenario	1. User presses "start chat with financial adviser" button 2. Financial adviser is notified of the request 3. User is connected with a financial adviser
Failure Scenarios	3a. User cannot connect with financial adviser due to network problems - 3a1. An error message is displayed telling user of the error, and potential solutions - 3a2. The app prompts the user to try again after a set time period (e.g. 30s).

See lecture notes "W3 L1 RE" (slides 46-47) and "W3 L2 More on RE" (slides 8-9) for examples of how to specify requirements correctly

2a. No courses are available for this student.

— 2a1. System displays error message saying no courses are available, and provides the reason & how to rectify if possible.

— 2a2. Student backs out of this use case and tries again after rectifying the cause.

5a. Some courses could not be registered.

— 5a1. System displays message showing which courses were registered, and which courses were not registered along with a reason for each failure.

5b. None of the courses could be registered.

— 5b1. System displays message saying none of the courses could be registered, along with a reason for each failure.

See lecture notes "W3 L1 RE" (slides 46-47) and "W3 L2 More on RE" (slides 8-9) for examples of how to specify requirements correctly

Failure Scenarios

- It is not a list of things that can possibly go wrong

Failure Scenario(s):

- User registration fails due to a technical issue.
- Users cannot log in due to incorrect credentials.

What is the resolution?

- Failure scenarios are not bugs or negative cases in the app:

2. The user chose to be both the customer and restaurant owner

This looks like a bug/usability issue in the app!

- The user dislikes the recipes.

Negative scenario but still expected (success not error)

Non-functional Requirements (NFR)

- Should be measurable and verifiable
- Should be project-specific (not generic sentences which can apply to any project)

Non-Functional Requirements:

1. Performance is crucial to ensure a smooth and responsive user experience, especially when generating itineraries and accessing recommendations. Performance can be validated by measuring the app's response time and ability to handle concurrent user requests under load.

- Should not use ”imaginary” / not justified numbers (search online for realistic expectations)

Content should be rendered within 30 seconds

- Should be realistic (you will need to make it work!

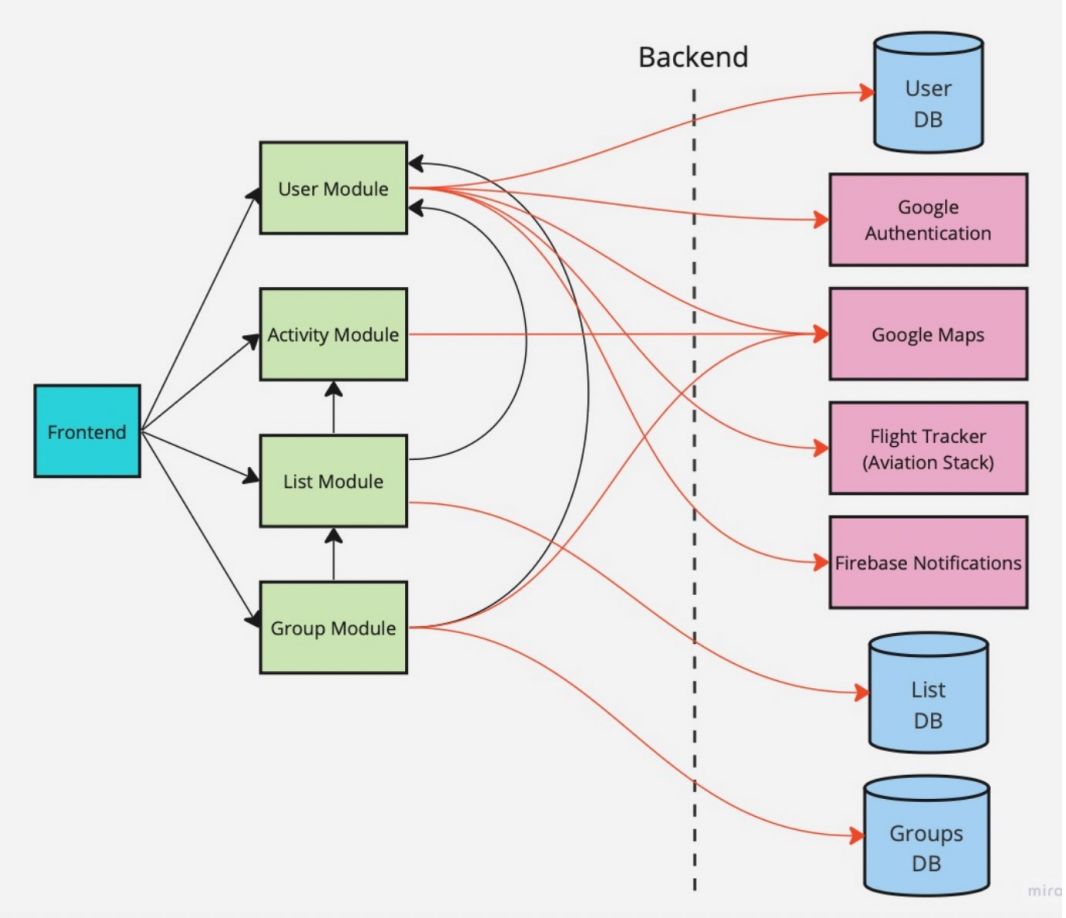
d. The server should be available 24/7 to the users with no to minimal downtime

Design Common Questions/Mistakes

- No separation of concerns / not following single responsibility principle
  - Users =/= Food, cannot be handled together
- What if XXX is just a database and we do not need any module?
  - There should be a module that wraps database requests
- What if a module just updates the database and has no interfaces?
  - There should be other modules that require this data.
  - They should not access the database directly (single responsibility principle, the system is harder to secure if all modules access the database directly).
- Overcomplicating
  - In SE, simple, clear, and correct is good
    - Make sure to satisfy the requirements
    - **.. But if something is not required, no need to include it.**

Design Warnings!

- Database separation is clean but…
- APIs are accessed by too many modules
- Modules access too many external components (User module: maps, flights, notifications)



Exercise: Restaurant Management System

You are building a food delivery system, where customers can browse and select food items, pay, and order delivery.

1. Describe the main actors and use cases of your system. Draw a use case diagram.

- Main actors:
  - Customer
  - Payment company
  - Delivery company

- Use case:
  1. Browse food items
  2. Manage cart
  3. Place and pay for the order
  4. Order delivery

Exercise: Restaurant Management System

You are building a food delivery system, where customers can browse and select food items, pay, and order delivery.

2. Which modules will be part of your backend?

Exercise: Restaurant Management System

You are building a food delivery system, where customers can browse and select food items, pay, and order delivery.

2. Which modules will be part of your backend?

- Menu (to manage food and interact with food dataset)
- Orders (to manage shopping cart and make an order)
- Payments (to manage payments and interact with payment company)
- Delivery (to manage and track deliveries and interact with shipping company)

Exercise:
Restaurant Management System

Consider the following scenario:

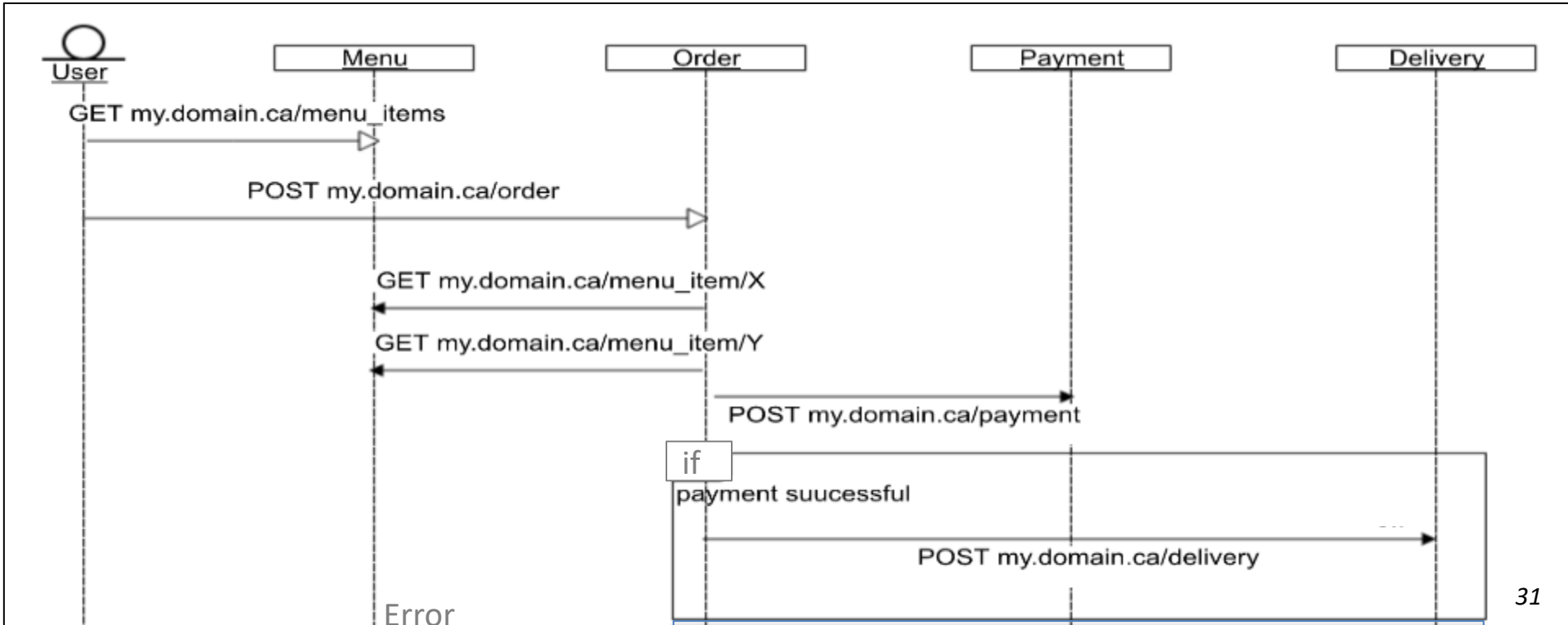
- *a customer browses the menu,*
- *picks items X and Y,*
- *pays with their credit card C,*
- *and orders delivery to the home address A.*

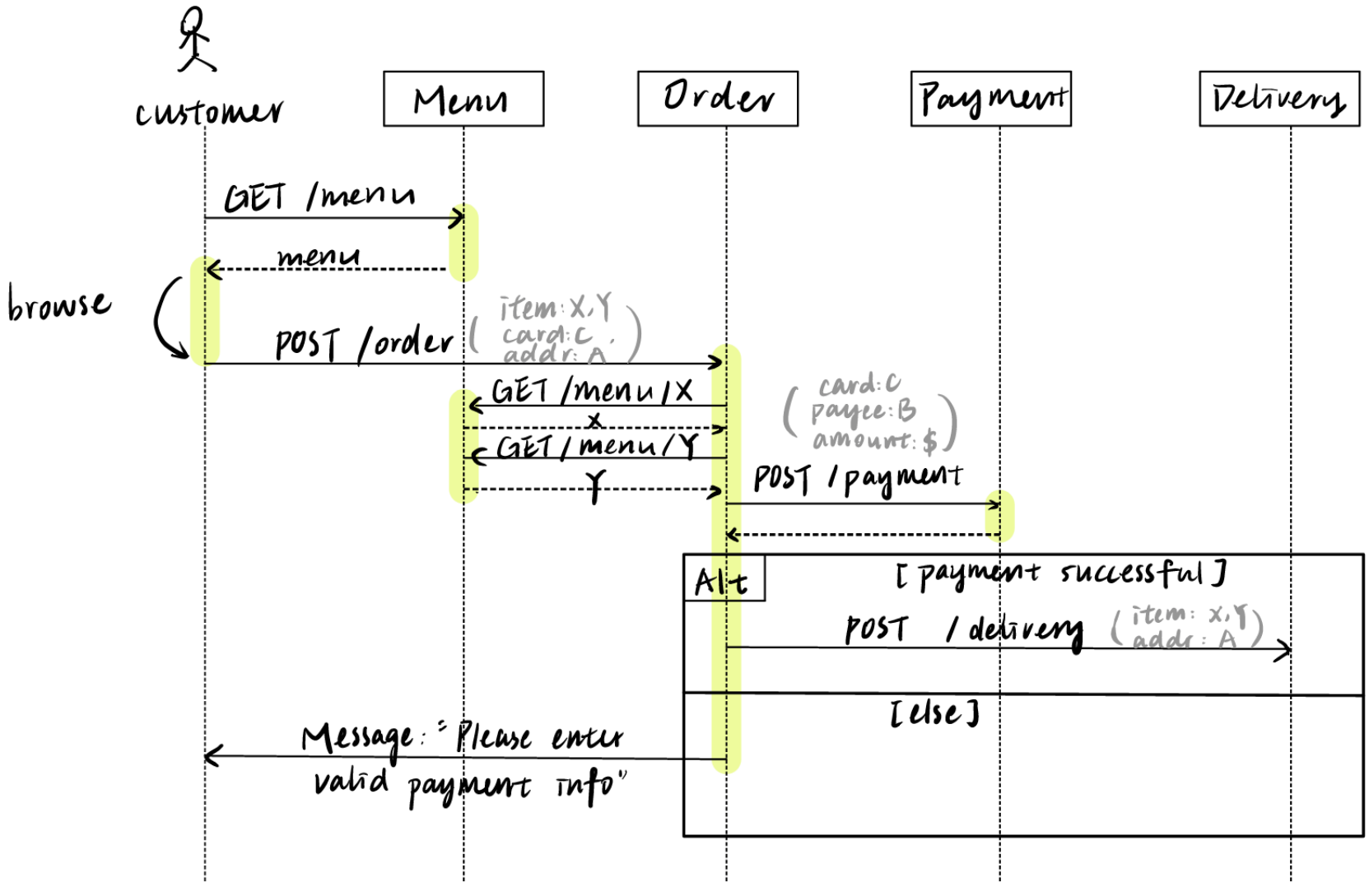
3. **Identify main REST APIs needed to implement this scenario and draw a sequence diagram to show how these APIs will be used to implement the scenario.**

<u>Reminder:</u>

GET: Retrieve a representation of a resource.
POST: Create a new resource.
PUT: Update a resource (existing URI).
DELETE: Clear a resource

- Menu
  GET my.domain.ca/menu_items
  GET my.domain.ca/menu_item/id
- Order
  POST my.domain.ca/order
  items: X, Y; credit info: C;
  delivery address: A
- Payments
  POST my.domain.ca/payment
  amount: XXX; credit info: C
- Delivery
  POST my.domain.ca/delivery
  items: X, Y;
  delivery address: A




customer
Menu
Order
Payment
Delivery
GET /menu
menu
browse
POST /order
(item: X, Y
card: C
addr: A)
GET /menu/X
X
GET /menu/Y
Y
(card: C
payee: B
amount: $)
POST /payment
Alt
[payment successful]
POST /delivery
(item: X, Y
addr: A)
[else]
Message: "Please enter
valid payment info"

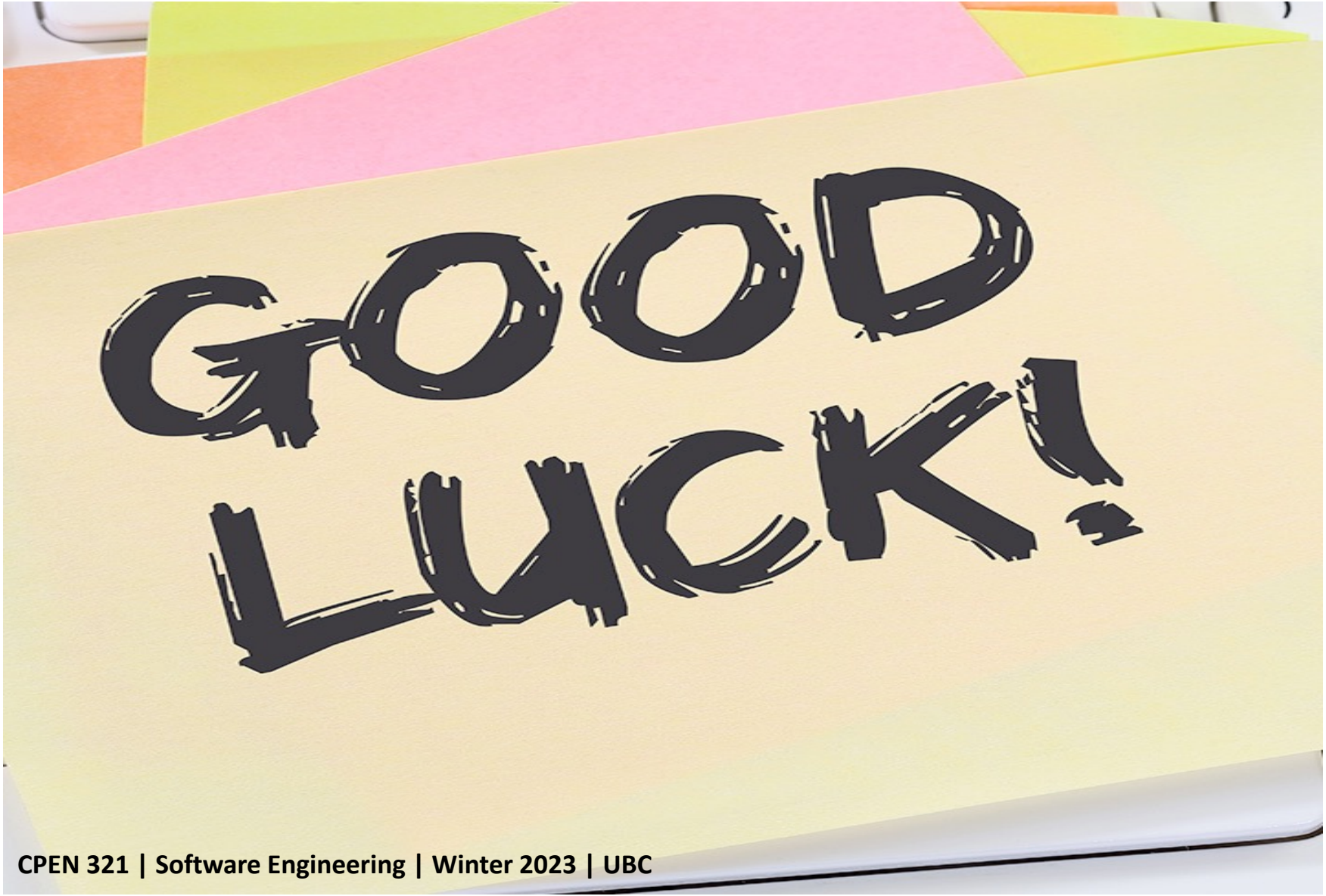
Exercise: Main Points

1. All objects (use cases, classes) must have descriptive names (+ need to include textual explanations when their role is not obvious from the context)
2. Messages should be labeled with appropriate interfaces
3. Include success and fail paths
4. Encapsulation and levels of abstraction: users should not typically talk directly to databases. Use the MVC pattern instead.
5. Focus on interactions between the main components for accomplishing each task (interfaces)
6. Make sure the information flows between components rather than coming out of nowhere
   - If you need to search by ID, the ID should be retrieved first

Questions?

Agenda for the Rest of Today

- Some common mistakes
- A design exercise
- M4 (MVP) spec
  - Posted on Canvas
  - No major surprises (all functional requirements implemented and working)
  - Will require updated requirements and design (if changed from M2-M3)
  - Will require a report on
    whether and how ChatGPT 3.5 was used to help with this assignment
    or
    what alternatives were used (Stackoverflow or similar)
  - **No other AI technologies are permitted** to help with the implementation effort!


GOOD
LUCK!