UML

Software Engineering is a discipline whose goal is the **production of quality software**, **delivered on time** and **within budget**, that **satisfies the user's needs**.

It encompasses **processes** (project planning, requirement development), **management techniques** (monitor progress, control changes), and **technical methods** (requirement gathering techniques, OOAD, testing, coding).

Software Development Life Cycle (SDLC) is the process of building, deploying, using, and maintaining a system.

1. **Planning** - identify scope; ensure feasibility
2. **Analysis** - document business needs; process requirements of system
3. **Design** - design solution system
4. **Implementation** - build, test and install reliable information system
5. **Support** - keep system running for many years of system's lifetime

Software Engineering Methodology defines the approach or framework used during software development.

- Waterfall
- V-shaped
- Spiral
- Incremental
- Extreme programming (Agile development)
- DevOps

Unified Process (Agile, Scrum) - an iterative and incremental approach

1. **Inception** - develop system vision; define scope; rough estimates for cost and schedule
2. **Elaboration [Iteration]** - refine vision; describe requirements; finalize design; implement core architecture and functions
3. **Construction [Phase]** - iteratively implement remaining lower-risk elements
4. **Transition** - complete beta test and deployment

Each **iteration** is like a mini project and it involves activities from all 9 disciplines. **Discipline** is a set of functionally related activities which produce artifacts.

Main Development Disciplines - used for specific duration

1. Business Modeling
2. Requirements
3. Design
4. Implementation
5. Testing
6. Deployment

Support Disciplines - used throughout the duration

7. Project Management
8. Configuration and Change Management
9. Environment

Activities of the Requirements Discipline

- Gather detailed information
    - Conduct interviews and discussions with the users
    - Building effective prototypes (discovery, evolving, mockup)
    - Distribute and collect questionnaires
    - Review reports, forms, procedure, descriptions
    - Observe business processes
    - Conduct Joint Application Design sessions (JAD)
    - Research vendor solutions

Iteration is required for information gathering due to incomplete information in the initial phase. There might be conflicting views or overlapping requirements among stakeholders, hence iterations create a feedback loop between stakeholder, developers, and users. Iterations allow for clarification and refinement of the requirements, leading to a more successful project outcome.

- Define **functional requirements** - activities or processes that system must perform and are essential for successful software development to meet user needs

- Define **non-functional requirements** - essential characteristics that describe how a software system should perform, rather than specifying specific functionalities; address quality attributes and shapes overall user experience and system behavior; constraints on the system (performance, usability, reliability, security)
- Prioritize requirements
- Develop user interface dialogs
- Evaluate requirements with user

Software Modeling documents the different parts of a problem and solution, show different levels of system details, and is a tool for communication among stakeholders of a project.

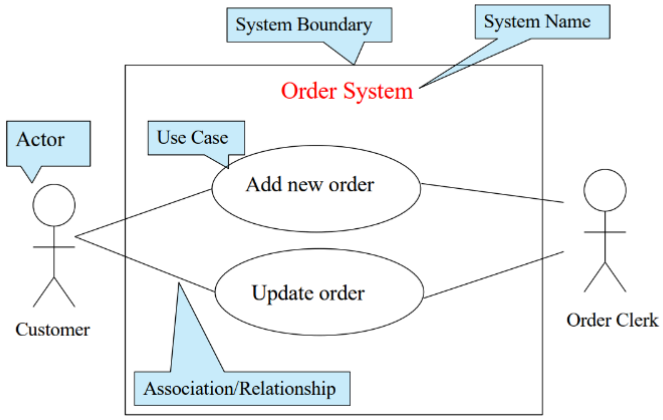
Software Models reduce complexity of components to essentials, act as visual cues, allow to clarify and refine requirements early in the development, provide a way to store information as documentation, allow for effective teamwork amongst project team members, and promote informal training.

Unified Modeling Language (UML) is a pictorial language used to make software blueprints.

Stereotype in a UML model means an extension of the vocabulary of UML.

UML Diagrams
- Use case
- Class
- Sequence
- Communication
- Statechart
- Package
- Deployment

Use Case Diagram - to illustrate the interactions between the system and external entities

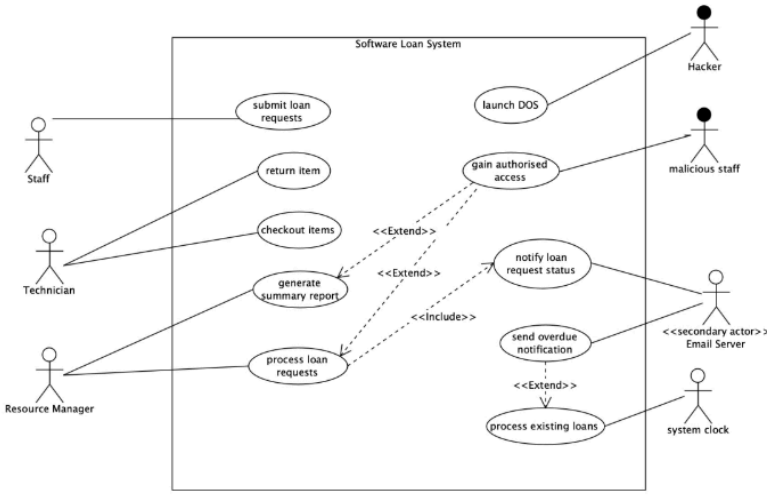


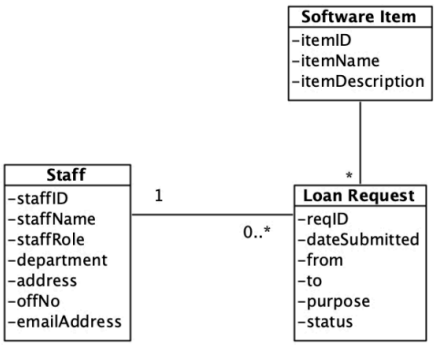
Scenarios represent possible situations that may arise. System must be able to handle all these scenarios.

Misuse cases are scenarios that describe how a system can be abused or attacked. They help identify potential security threats and vulnerabilities that need to be addressed to protect the system.

<< include >> is mandatory
<< extend >> is optional (points to base use case)

E.g. Global Knowledge Pte Ltd has a resource center to loan the software items to its internal staff within the company. A web application is to be developed with the functions described below. The system allows staff to submit loan requests by specifying the purpose and the duration of the software items to borrow. The system will route the requests to the resource manager for approval and the staff will be notified by email on the outcome of the request. The system will allow the staff to check-out the items with a loan transaction created. The software must be returned by the due date. The return date will be recorded, and the loan record status will be updated. If the loan is overdue, a reminder email will be sent to the relevant staff. The system also allows the Resource Manager to generate a summary report to consolidate all the loan transactions at any point in time. As far as the intended performance is concerned, the web application should support online loan applications for 50 concurrent users with processing time not more than 10 seconds in each request.





Use Case Description

Eg1.

- Use Case ID: SLS.UC1.0
- Use Case Name: **submit loan requests**
- Description: This use case is used by any staff in the organization to put up request to the IT department to borrow any organizational software for installation on the staff work PC. To complete the request, the staff need to specify the purpose and the duration of the software items to borrow. Once all the required information is submitted, the system will route the request(s) to the resource manager for approval. The staff will be notified by email on the outcome of their request. If their requests are approval, the staff will have to head to the IT department help to check out the software item.
- Primary Actor: Staff
- Secondary Actor: Resource Manager
- Precondition: Staff must login successfully
- Postconditions:
    1. Loan request created
    2. Request routed to Resource Manager
    3. loan request unsuccessful
- Main Flow
    1. The system retrieves list of software items
    2. The system displays list of software items
    3. The staff select software item
    4. The staff enters purpose for loan
    5. The staff enters return date for item
    6. The staff submit software loan request
    7. The system validates loan request
    8. The system creates new loan request

9. The system route request to Resource Manager
10. The use case ends

- Alternate Flow
    a. missing mandatory loan request information
        1. The system displays error message
        2. Use case resumes at main flow step 3
    b. Software item not available
        1. The system displays software unavailable message
        2. Use case ends

Eg2.

- Use Case ID: SLS.UC6.0
- Use Case Name: **process existing loans**
- Description: At a preset time, the system will retrieve all current active loan records from the system. The system will check all the current loan records to verify
- Primary Actor: system clock
- Precondition: batch processing must be created
- Postconditions:
    1. overdue loans notification sent
- Main Flow
    1. The system retrieves all current active loan records
    2. The system verifies due date of all loan records (Extension point: send overdue notification)
    3. The use case ends

Eg3.

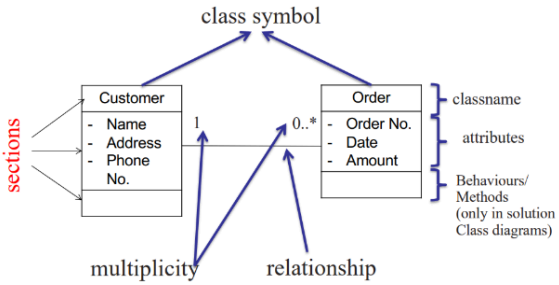
- Use Case ID: SLS.UC7.0
- Use Case Name: **send overdue notification**
- Description: If a staff software loan is overdue, an email notification will be sent to the relevant staff as a reminder for the staff to return the loan item. The system will have to retrieve the staff email address (so the email will be sent to the correct staff) as well as the details of the software that is on loan. The details of the software as well as the due date will be clearly stated in the email.
- Primary Actor: Email server
- Precondition: overdue loan
- Postconditions:
    1. Overdue loan email notification sent
- **Extends use case:**
    1. process existing loans
- Main Flow
    1. The system retrieves staff email address
    2. The system retrieves software loan details
    3. The system generates email message
    4. The system sends out email
    5. The use case ends

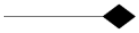
Problem Domain defines context and scope of the problem.

UML Class Diagrams - to model the static structure of a system, showing classes and their relationships at a particular point; object-oriented approach

- **Domain class** - models set of important data representation within a problem domain (tangible and intangible domain objects)
- **Solution class** - models objects within the system that has the capability to interact to either keep or track information of domain objects

(UML) Class Diagram Notation



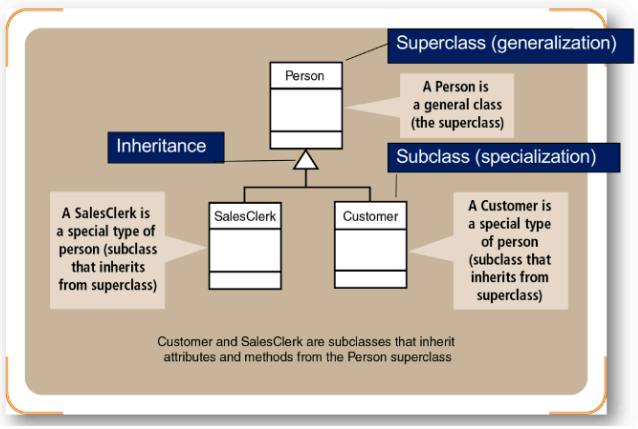
Relationship	Notation	Multiplicity
1. Association		
i. Bi-directional		✓
ii. Uni-directional		✓
2. Generalisation/Specialisation		
i. a.k.a Inheritance		X
ii. "Is a kind of"		
3. Aggregation		
i. "Consist of"		✓
4. Composition		
i. "made up of"		✓

Types of Class Relationships

Association

- In a many-to-many association, an association class is connected with a dashed line

Generalization/ Specification



Whole-part Hierarchy

- Aggregation - association with independent parts
- Composition - association with dependent parts

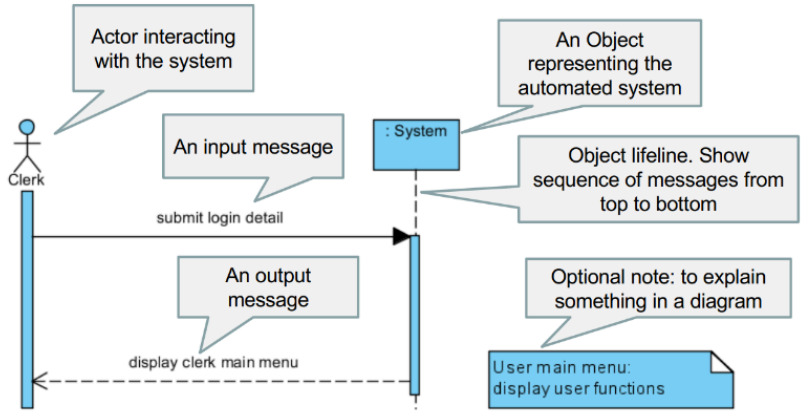
Object-oriented analysis and design (OOAD) - create blueprints or design models necessary to build systems; models created in OOA are refined and extended in OOD to produce systems design

OOA - analyzing problem domain to develop conceptual models (use cases, UML domain class diagrams, interaction diagrams, user interface mock-up)

OOD - implementation constraints are applied to conceptual models to derive a solution model

Use case realization means to adopt OOAD techniques to further develop use cases, making them **executable**.

System Sequence Diagram (SSD) - gives a very high-level view of the interactions between system and the actors

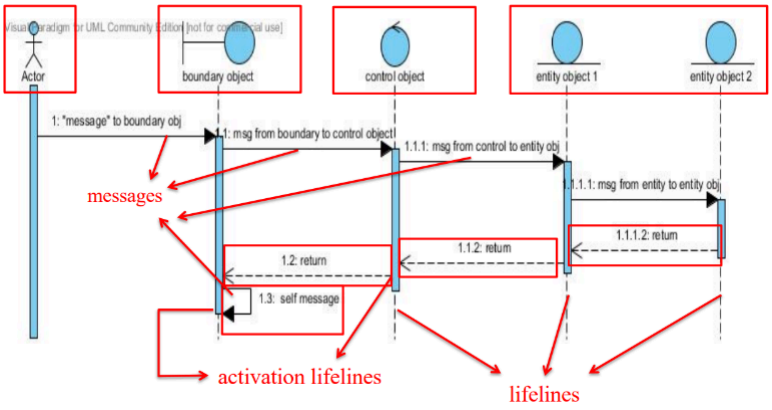


Components of a SSD

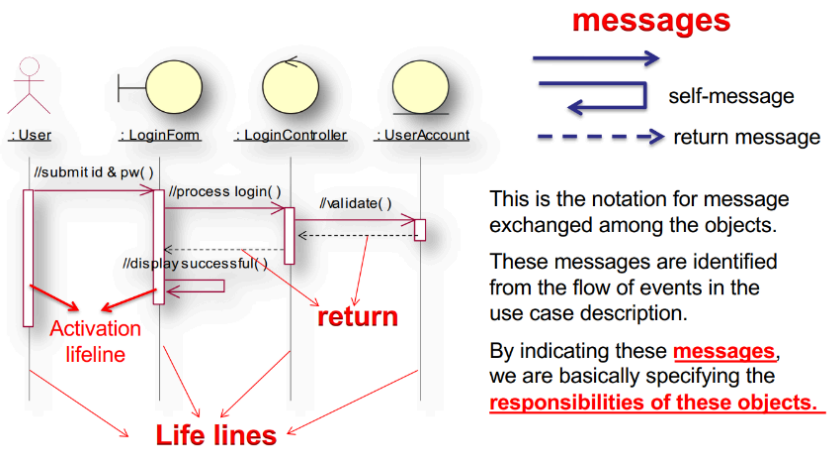
Sequence Diagram - to represent the dynamic behavior of a system, focusing on the flow of messages between objects; describe the sequence of events that occur during a particular use case

Requires use case model + use case description + domain model.

The :System object is replaced by internal objects and messages within the system.

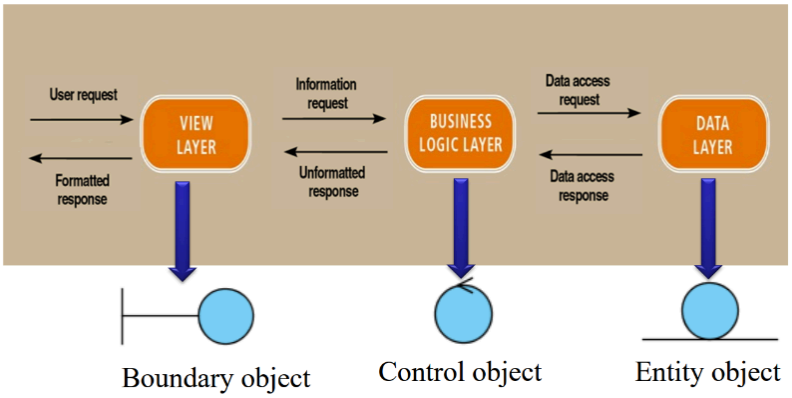


Eg1. Login Use Case



3-layer architecture

Allows for better scalability, performance, and easier maintenance.



Data Layer - storage and retrieval of data
Eg. stores/ handles the data about login id and password
Suitable entity classes can be identified from the domain model. Insert, retrieve, update and delete.
LoginAccount is the <<entity>>

Business Logic Layer - processes business logic, rules and computations
Eg. handles the process of validating the login
Every use case should have a control class to coordinate the activities.
LoginController is the <<control>>

View (Presentation) Layer - found on client side, handling user interface
Eg. allows users the means to enter id and password
The boundary class represents the user interface form. Formats and displays responses to the user.
LoginForm is the <<boundary>>

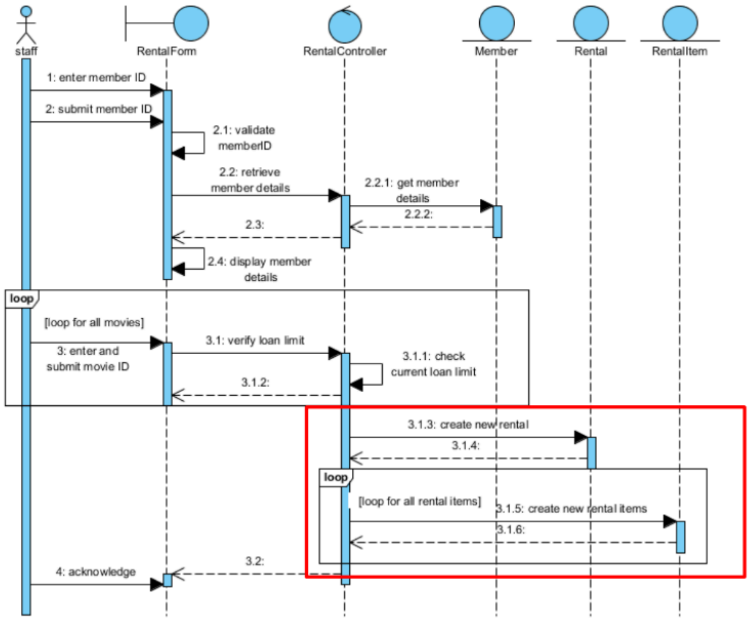
Advantages

- Separation of concerns - minimize overlap in functionality
- Code reuse
- Easy to implement changes
- Enable parallel development of the different tiers of the application
- Improved data integrity
- Improved security

Disadvantages

- More complex structure
- Increased points of communication
- Lack of persistent database connection
- Proxy server requirement

Eg2. Rent Video Use Case



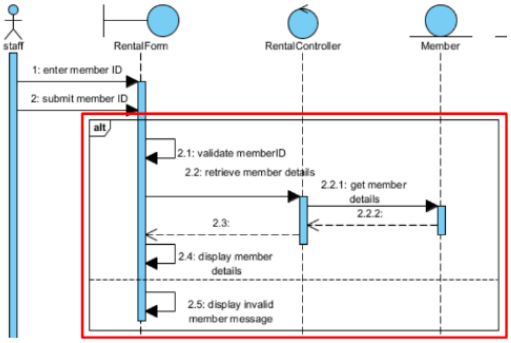
Alternate flow

Alternate Flow

2a. Invalid member

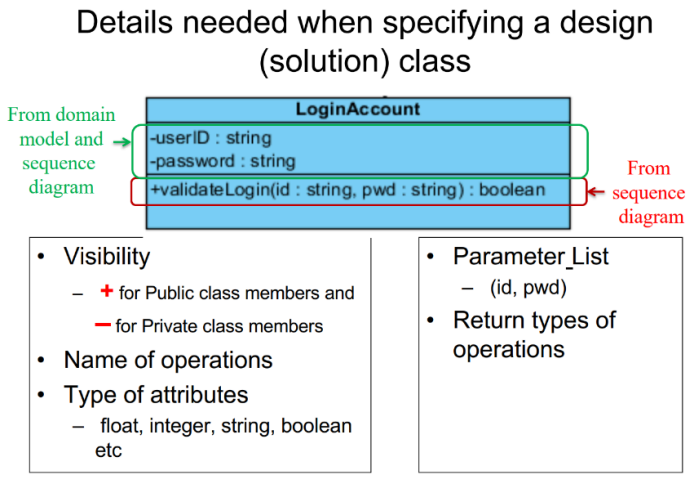
i. The system prompt "Invalid Member" error message

ii. Use case ends

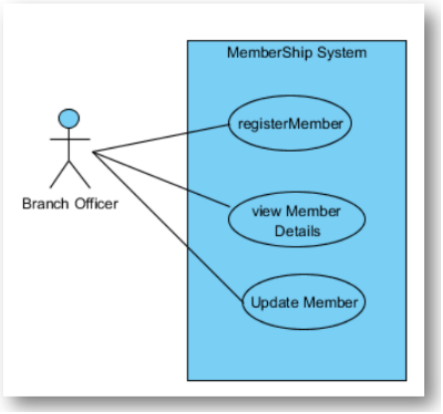


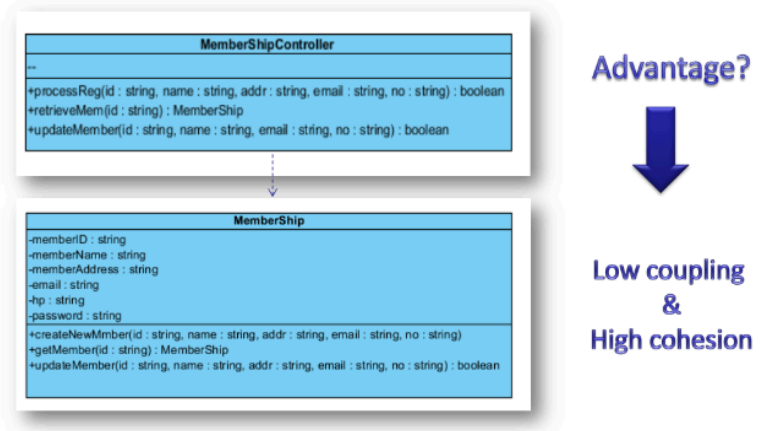
UML Design Model/ Solution Class

- Converts the responsibilities of the analysis objects into operations (methods)
- High cohesion, Low coupling



Eg. MemberShip System



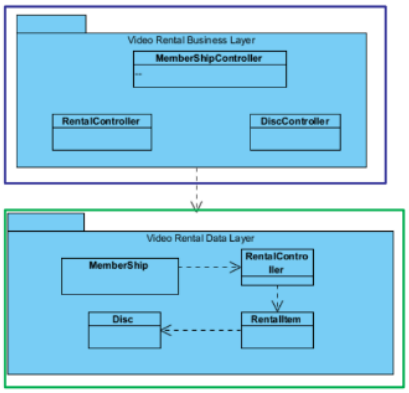


This improvement also helps to achieve high cohesion for the Membership controller class as we focus all membership operations in this class.

Package Diagram

Grouping related classes into packages based on functionality and present the overall system design as subsystems. A Package is a general purpose mechanism for organizing model elements & diagrams into groups.

Example: Package diagram for a Disc Rental System



We could have 2 packages to group these classes into:

Business Logic Layer (or subsystem) that contains:

- MembershipController
- RentalController, DiscController

Entity (Data) Layer (or subsystem) that contains:

- Membership
- Disc
- Rental
- RentalItem

This is another way to represent the overall class diagram in design

For each use case, convert the responsibilities into methods for controller and entity objects. Consolidate into overall controller and entity solution classes.

Basics of Web Programming

Front-end

1. Hyper Text Markup Language (HTML)
   A language to define the content and structure of a web document. Each element consists of an open tag, a close tag and the children.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>50.003</title>
    <link rel="stylesheet" href="hello.css" />
  </head>
  <body>
    <p class="heading">Welcome to 50.003!</p>
    <p id="msg1" class="message">It's about Software Engineering!</p>
  </body>
</html>
```

Structure defined by this HTML file ->
* html
  * head
    * meta (specify the character set, page description, keywords, author of the document, and viewport settings)
    * title (required)
      * text
    * link (css file)
  * body
    * paragraph
      * text

2. Cascading Style Sheets (CSS)
   Focuses on the aesthetical requirement of the document.

```
selectors { (declaration) property: value; }
```

Note: Selectors are composable, e.g. selector p.heading selects all <p> elements with class equal to heading.

```
p {
  margin: 10px;
}

#msg1 {
  font-family: "Courier New", Courier, monospace;
}

.heading {
  font-size: larger;
}
```

3. JavaScript
   Include a <script> element. Note that we can't use <script/> short-hand as it won't work in certain browsers.

```
<script src="hello.js"></script>
```

Content of the hello.js file:

```
console.log("hello");
```

Note that the printed string will not appear in the HTML page, instead it is printed in the console of the browser. To display "hello" message in a pop-up window, we could change the console.log() method with alert(). To add a string to the beginning of the HTML document, we may use document.write() method.

```
var msg = "hello";
/* console.log(msg);
alert(msg); */
document.write(msg);
```

Back-end

1. Database
2. Back-end application server
3. Cache storage (optional)
4. Load balancers (optional)
