**QUESTION 1**

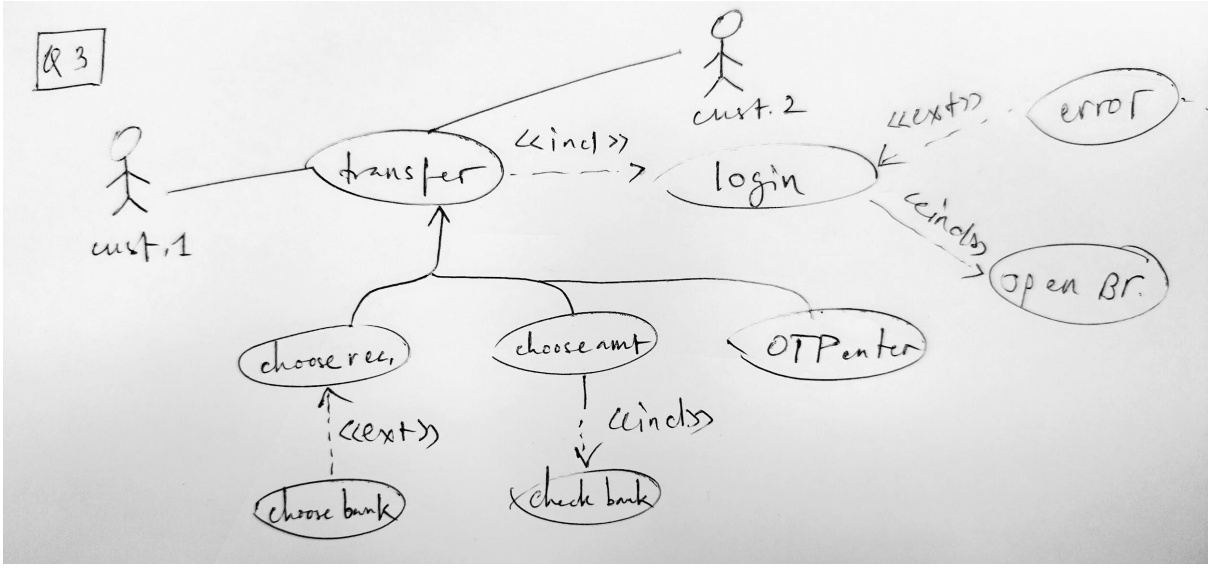
1. Requirements Phase:
- Objective: Gather and document all the requirements of the software product to be developed. This involves thorough communication with stakeholders to understand their needs and expectations.
- Activities: Requirement elicitation, analysis, documentation, and validation. Creation of a detailed requirements specification document.
2. Design Phase:
- Objective: Develop a comprehensive design based on the gathered requirements. The design phase focuses on creating the blueprint of the software system, describing its architecture, and defining data structures and algorithms.
- Activities: Architectural design, low-level design, user interface design, database design, and system flow representation.
3. Implementation Phase:
- Objective: Convert the design into actual code. The implementation phase involves writing, coding, and unit testing the software components based on the detailed design specifications.
- Activities: Writing code, code integration, unit testing, and fixing defects.
4. Testing Phase:
- Objective: Verify and validate the developed software against the specified requirements. This phase ensures that the software performs as intended and meets the user's expectations.
- Activities: System testing, integration testing, acceptance testing, and bug fixing.
5. Deployment Phase:
- Objective: Release the software to end-users or customers. This phase involves deploying the software to the production environment and making it available for actual use.
- Activities: Software deployment, user training, user acceptance testing, and ongoing support.

The modified waterfall model allows feedback and adjustments at each phase, which helps in catching errors and addressing changes early in the development process.

**QUESTION 2**

1. What are the most important features that you want in the web-based banking app?
2. What are the security requirements for the web-based banking app?
3. What are the performance requirements for the web-based banking app?
4. What are the user interface requirements for the web-based banking app?
5. What are the compliance requirements for the web-based banking app?

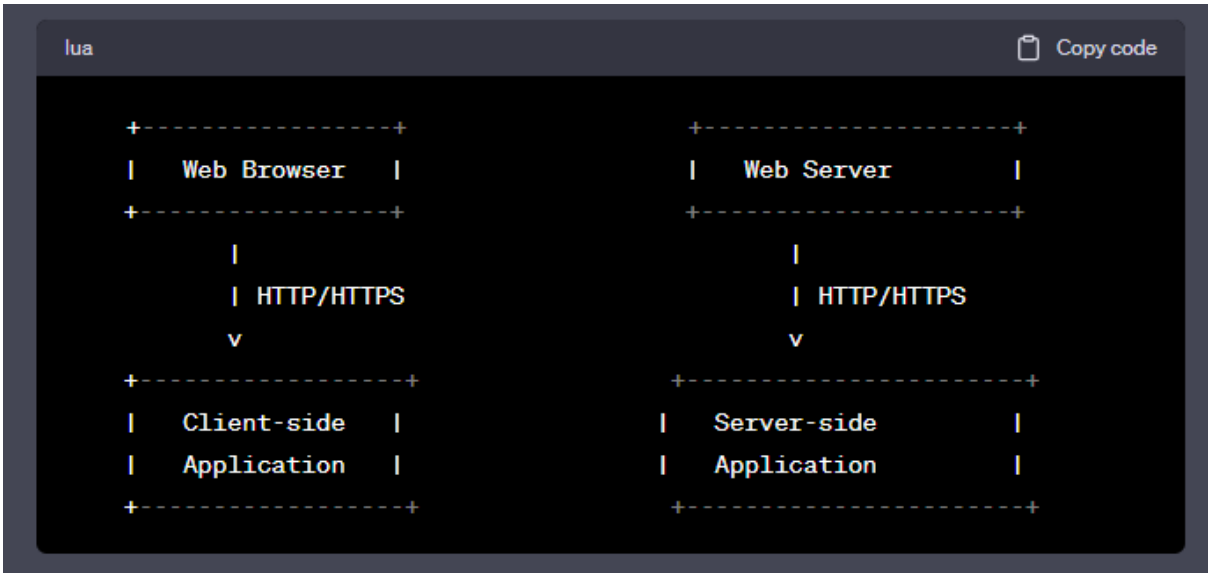
**QUESTION 3**




**QUESTION 4**

| Requirement | Priority | Reason |
| --- | --- | --- |
| Security | Must-have | Security is the most important requirement for a web-based banking app. It is important to ensure that the app is secure and that users' data is protected. |
| User Interface | Must-have | The user interface is also an important requirement for a web-based banking app. It should be easy to use and navigate, and should provide users with all the information they need. |
| Compliance | Should-have | Compliance is an important requirement for a web-based banking app, but it is not as critical as security or user interface. The app should comply with all relevant regulations and standards. |
| Performance | Could-have | Performance is also an important requirement for a web-based banking app, but it is not as critical as security or user interface. The app should be fast and responsive, but this is not as important as ensuring that it is secure and easy to use. |
| Features | Could-have | Features are also an important requirement for a web-based banking app, but they are not as critical as security or user interface. The app should provide users with all the features they need, but this is not as important as ensuring that it is secure and easy to use. |

**QUESTION 5**




**QUESTION 6**

An IDE stands for Integrated Development Environment. It is a software application that provides facilities such as a source code editor, build automation tools and a debugger to programmers for software development. It allows developers to start programming new applications quickly because multiple utilities don't need to be manually configured and integrated as part of the setup process. Developers also don't need to spend hours individually learning how to use different tools when every utility is represented in the same workbench.

**QUESTION 7**

1. Identify the Reasons for Delays: Before taking any action, it's essential to identify the root causes of the delays. Review the project plan, timeline, and progress to pinpoint the factors responsible for the lateness.
2. Assess Impact and Prioritize: Evaluate the consequences of the project delay. Determine if the delay affects the overall project goals, business objectives, customer commitments, or any other critical factors. Based on the impact, prioritize the remaining tasks and requirements.
3. Review Scope and Requirements: If the project is significantly delayed, it might be necessary to reevaluate the project scope and requirements. Identify any unnecessary or non-critical features that can be deferred to a later release to expedite the current delivery.
4. Communicate with Stakeholders: Transparent and honest communication with stakeholders is crucial. Inform them about the delay, the reasons behind it, and the proposed plan to get the project back on track.
5. Rework Project Plan: Reassess the project plan, timeline, and resource allocation. Set new realistic milestones and deadlines based on the current progress and remaining work.
6. Avoid Overtime as the First Solution: While asking current developers to work extra time might seem like a quick fix, it should not be the default solution. Prolonged overtime can lead to burnout, reduced productivity, and compromised code quality.
7. Allocate Additional Resources Wisely: If the current team is unable to meet the revised timeline, consider hiring more developers or bringing in additional resources. However, it is essential to onboard new team members carefully, ensuring they have the required skills and knowledge to contribute effectively.

8. Agile Approach: If the project is suitable for an Agile development methodology, consider transitioning to it. Agile allows for incremental and iterative development, which can help deliver valuable features faster and adapt to changing requirements.
9. Risk Mitigation: Evaluate potential risks and uncertainties in the project and implement strategies to mitigate them. Addressing risks early can prevent further delays.
10. Learn from Mistakes: After successfully resolving the late project, conduct a post-mortem analysis to identify lessons learned and ways to avoid similar issues in future projects.