Task 1:

Week 1 Meeting
Date: Tuesday, 24 September 2024
Time: 17:00 PM
Location: Library Meeting Room 2nd Floor

Attendees:

- **Laith Yaser** (CloudTables-Customer)
- **Salah Samara** (CloudTables-Service)
- **Hussein** (CloudTables-Manager)
- **Osama** (CloudTables-Operation)

Agenda:

1. Introduction and team formation.
2. Discuss coursework structure and deliverables.
3. Division of tasks among team members.

Meeting Minutes:

- The team members introduced themselves and agreed to meet every other Tuesday.
- Task assignments were finalised:
  - **Laith Yaser**: CloudTables-Customer subsystem.
  - **Salah Samara**: CloudTables-Service subsystem.
  - **Hussein**: CloudTables-Manager subsystem.
  - **Osama**: CloudTables-Operation subsystem.
- Team collaboration tools (GitHub) were set up.
- Next steps:
  - Members will begin analysing the case study and requirements in preparation for Task 2.

Week 3 Meeting
Date: Tuesday, 8 October 2024
Time: 17:00 PM
Location: Library Meeting Room 2nd Floor

Attendees:

- **Laith Yaser** (CloudTables-Customer)
- **Salah Samara** (CloudTables-Service)
- **Hussein** (CloudTables-Manager)
- **Osama** (CloudTables-Operation)

Agenda:

1. Progress update on initial research.
2. Plan and start Task 2.
3. Set deadlines for Task 2.

Meeting Minutes:

- Each member shared their understanding of the subsystem requirements.
- Members agreed to focus on defining quality attributes (security, performance, reliability, scalability) for Task 2.
- Deadline for completing Task 2: **22 October 2024** (next meeting).
- Members will prepare drafts of their quality requirements and bring them for review in the next meeting.

Week 5 Meeting
Date: Tuesday, 22 October 2024
Time: 17:00 PM
Location: Library Meeting Room 2nd Floor

Attendees:

- **Laith Yaser** (CloudTables-Customer)
- **Salah Samara** (CloudTables-Service)
- **Hussein** (CloudTables-Manager)
- **Osama** (CloudTables-Operation)

Agenda:

1. Review Task 2 quality requirements for all subsystems.
2. Begin Task 3 (Use Case and Activity Diagrams).
3. Plan the next steps.

Meeting Minutes:

- Task 2 drafts were reviewed. Members cross-checked each other's work to ensure alignment and consistency.
- Members started working on Task 3:
    - Each member agreed to develop a Use Case Diagram and an Activity Diagram for their subsystem.
- Members agreed to complete Task 3 by **5 November 2024** (next meeting).

Week 7 Meeting
Date: Tuesday, 5 November 2024
Time: 17:00 PM
Location: Library Meeting Room 2nd Floor

Attendees:

- **Laith Yaser** (CloudTables-Customer)
- **Salah Samara** (CloudTables-Service)
- **Hussein** (CloudTables-Manager)
- **Osama** (CloudTables-Operation)

Agenda:

1. Review completed Task 3 diagrams.
2. Begin Task 4a (Subsystem Architecture Design).
3. Plan for Task 4b and Task 5.

Meeting Minutes:

- Task 3 (Use Case and Activity Diagrams) was reviewed and finalised.
- Members began Task 4a:
    - Agreed to focus on architectural design, specifying microservices, and preparing UML component diagrams.
    - Set the deadline for completing Task 4a: **19 November 2024** (next meeting).
- Members started discussing ideas for Task 5 (Detailed Design).

Task 2:

(a) *CloudTables-Customer*: a mobile app for running on smart phones for restaurant customers to make table booking, food order, bill payment, and service review and ranking, etc.

Security

-Security And Privacy

- Data Encryption : CloudTables-Customer encrypts customer data at the application level—booking data, credit card info, ratings, etc.—and transmits it to the cloud so that if someone attempts to access info in transit—or on the server—it's unreadable. More specifically, the application uses AES-256 for data encryption and RSA for key exchange between the application and cloud. Therefore, the strength of encryption is strong, yet the user experience is seamless and safe.

-Decentralised Identity Management

- Decentralised Identifiers (DIDs): Implement Decentralised Identifiers (DIDs) to allow users to create and control their identities. This is a preferable option to having identity characteristics stored in a central location, which is susceptible to being hacked in a data breach affecting millions. Self-Sovereign Identity

- (SSI): Implement self-sovereign identity capabilities for users. Users decide which information needs to be disclosed. For instance, a person can prove that they are over 18 without disclosing their date of birth; only the identifying characteristics will be displayed.

-Privacy-Preserving Analytics

- Homomorphic Encryption: Employ homomorphic encryption to be able to compute on ciphertext so that the system learns things (like when patrons book most often) without ever seeing decrypted patron information. This means analytics and feedback are produced without compromising privacy and security.
- Federated Learning: Implement federated learning when analysing patron usage patterns to enhance app functionality. That way, the computing happens without ever leaving the patrons' devices, and only the aggregate results come back to the main server for enhancement. This means no individual patron's information ever leaves their device—which is privacy and security beyond belief.

Performance

-Edge Computing For Latency Reduction

- Edge Computing: Instead of sending data all the way to the cloud to get processed, it gets processed where it needs to be processed, closer to the end user devices. Edge nodes in particular geographic locations allow for the processing of updating table availability or suggested menu options to occur at the edge so that it is faster in response time and loaded off the cloud.
- Edge Caching: Data that is needed frequently—menus for restaurants, busiest times for reservations—is cached at edge nodes so that latency is decreased for those users who need it frequently.

-Granular Data Partitioning

- Partition Data Based on Booking Types: Partition data in different reservation types, such as reservations for dine-in, take away, and private events; so that the querying isn't as load heavy and retrieval can happen faster without needing to go into a backed up single table for all bookings.
- Column Storage for Frequent Queries: Use column storage for data that's frequently queried—what do people order most, what time of day are bookings the most—so that analytical queries are easier and faster. Column storage is best suited for operations with a high read rate and allows for insights to be had quicker.

-Optimised Mobile Experience

- Device Resource Adaptation: Utilise mobile hardware acceleration and change expectations based on the device. For instance, stronger and newer devices can use more powerful graphics and transitions while weaker devices are given a simplistic UI to prevent overexerting any one hardware capability while facilitating effective functioning across the application.
- Offline Functionality with Local Caching: Users can see past purchases and items in a shopping cart currently without opening the app when online. By caching this relevant information locally, access is almost instantaneous—even with Wi-Fi off. However, once the device powers back on, the app readjusts to sync with the cloud.

Reliability

-Blockchain For Distributed Reliability

- Immutable Customer Review: Store customer feedback in the form of service reviews using blockchain for authenticity and reliability of user information. It will avoid fraudulent modification and preserve the integrity of customer experiences recorded.
- Blockchain for Booking Ledger: A private blockchain could act as a distributed ledger for all bookings. Any transaction for booking takes place on-chain, forming it and rendering a decentralised means of accuracy and integrity across multiple nodes. This blockchain-based ledger can make it nearly impossible for booking information to be lost or tampered with, even if the central system experiences outages.

-Distributed Transaction Management

- Two-Phase Commit Protocol: Implement a two-phase commit protocol to ensure that distributed transactions (i.e., reserving a table, charging the customer, and sending the confirmation email) are properly completed and consistent across each microservice that needs to have such a transaction. This way, the customer payment goes through in every applicable microservice and the reservation is confirmed; or if something fails, neither goes through and the one table is available up for grabs with one in-progress potential option.
- SAGA Pattern for Long-Running Transactions: Implement the SAGA pattern for long-running transactions like the reservation itself. Reserving with an order will be broken down into microtransactions that can be compensated. If any of these microtransactions fail, compensating transactions will be executed to roll back any prior actions so that the system is still in a consistent state.

-Load Adaptive Redundancy

- Load Based Adaptive Replication: Implement a load-based adaptive replication approach which increases the number of replicas under increased load (i.e., when the system is busy and people are trying to book a reservation on a Saturday evening). This ensures read/write processes occur in a timely fashion.
- Hot Standby Replicas: Include hot standby replicas of the primary database which are instantly updated with the master instance and can take over failover instantly so that reservation or ordering activity is never offline.

<u>Scalability</u>

-Cloud Bursting For Dynamic-Scaling

- Hybrid Cloud Bursting: Augment capacity to a public cloud on a short-term basis as needed (weekends, busy holidays, etc.). While regular processing has resources/lives/calls in a private cloud because it's more cost-efficient, when traffic extends beyond what a private cloud can offer, everything in excess of the private cloud is "burst" into a public cloud like AWS or Azure to provide them with the Scalability without stressing resources
- Seamless Transition: Implement Proper data syncing between private and public clouds so that clients do not feel latency when bursting.

–Container Orchestration With Horizontal Pod AutoScaling

- Kubernetes for Orchestration: Deploying the application using Kubernetes will allow for horizontal pod autoscaling based on CPU and memory usage. If a traffic spike is detected, the system will automatically generate more pods to manage the load—with separate scaling for different subsystem components.

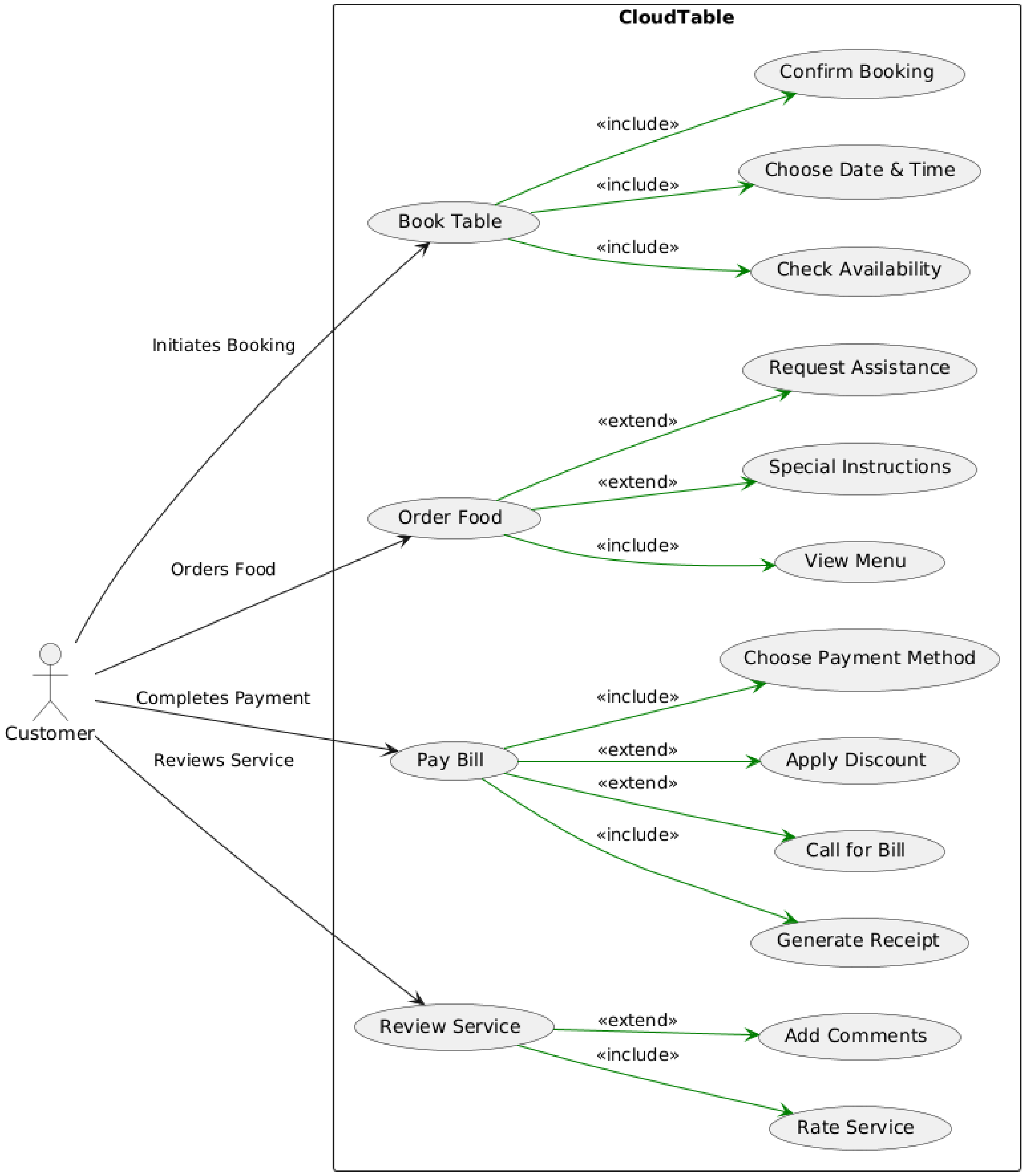
- Node Autoscaling: Kubernetes also provides autoscaling at the cluster level to join or leave cluster nodes so that during low traffic times, resources are not over-provisioned, but during high demand times, resources are not lacking.

-Containerized Microservices With Function Isolation

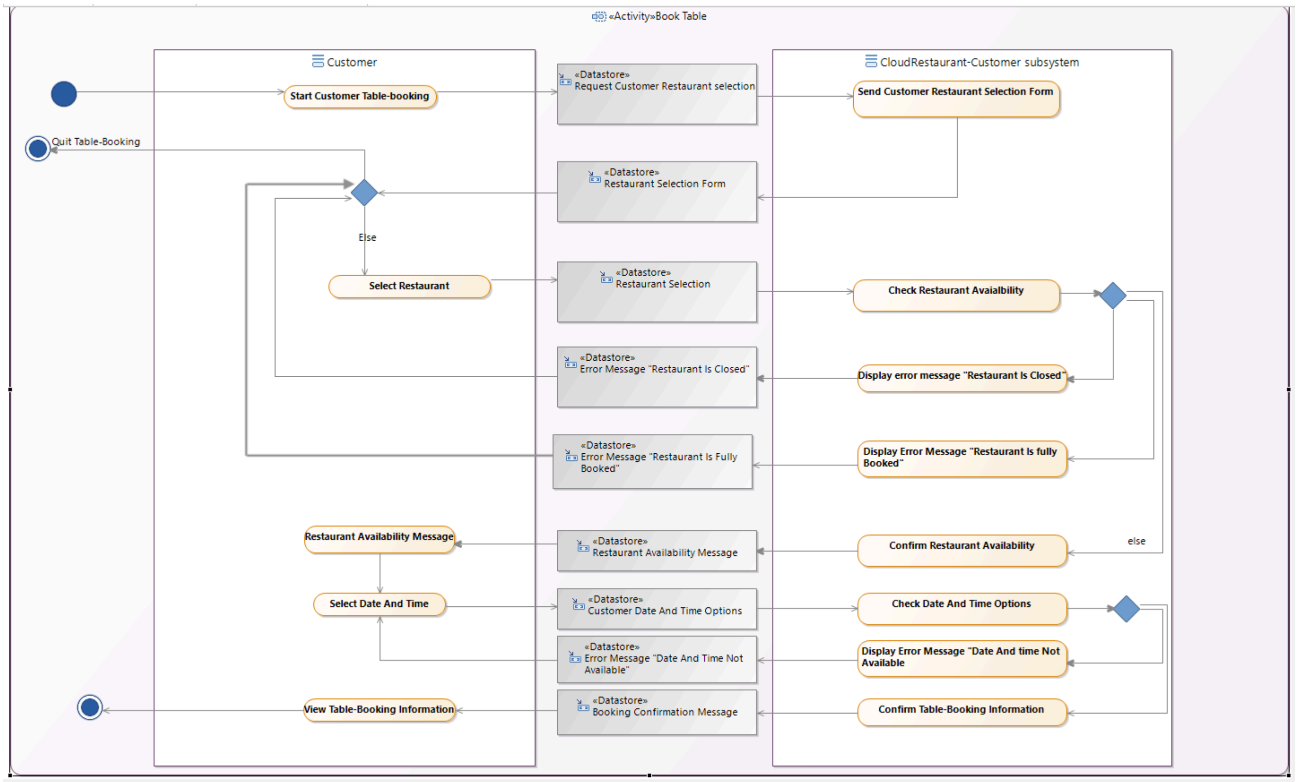
- Isolated Function Containers: Package each key function, for example table booking or ordering of food, in separated containerized microservices using light container platforms like Docker Swarm. This will ensure that the scaling of certain services is easily done when needed, without affecting other parts of the subsystem.
- Resource-Aware Scheduling: Make use of the resource-aware schedulers to route the workloads to the optimal node by utilising recent usages. It will scale the best at peak traffic, resource allocation, enhancing scalability and efficiency.

Task 3:

(a) *Use Case Model* (10 Marks, Individual effort): Each member of the team should develop one Use Case Diagram to define the use cases of the subsystem to specify the scope of the software engineering project.



(b) *Activity Model* (10 Marks, Individual effort): Each team member should select one use case of your subsystem to produce one Activity Diagram for the selected use case to specify the interactions between a user and the subsystem.



Task 4:

(a) *Architecture of the subsystem* (10 Marks, Individual effort): Each member of the team should produce an architectural design of your subsystem with focus on the microservices your subsystem provides and the microservices that your subsystem requests and other subsystems provides.