

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light green. They are positioned diagonally, with the blue one partially covering the green one.

# ENSF 607 Final Project Presentation

By: Momin, Satchy, Balkarn, Yajur



# Introduction

Best Appliances is a large home appliance chain that has numerous retail stores throughout Canada. Customers buy electronic equipment, and choose to purchase a support plan. Appliance experts will fix problems with the device at the customer's residence.

**The Task:** Provide a new architecture to improve the reliability of the application. Currently, the system is a large monolithic application which is prone to losing tickets and not accepting them. Whenever changes are made it results in bugs and failures.



# Brief Project Overview

We are revamping Best Appliances' outdated trouble ticket system with a new, resilient microservices architecture to address service reliability issues, improve system availability, and streamline change management. Our goal is to enhance customer satisfaction, reduce system failures, and ensure scalable and maintainable operations.

- 1) Defining the functional/non-functional requirements
- 2) Architectural Characteristics
- 3) Considering disintegrators/integrators
- 4) Choosing a deployment strategy
- 5) Utilizing fitness functions
- 6) Creating an architecture diagram



# Functional and Non-Functional Requirements

## Functional Requirements:

- Ticket System: Handle ticket creation, assignment, and tracking; prevent ticket loss.
- Expert Management: Manage expert profiles and match experts with relevant tickets based on credentials.
- Customer Interface: Allow customers to submit and track tickets.
- Billing System: Administer annual billing for support plans.

## Non-Functional Requirements:

- High uptime: ticket submission and expert assignment.
  - Handle increasing customer and ticket volumes.
  - Ensure tickets are not lost
- Facilitate easy updates and changes without affecting other components.
- Too many interconnected components cause crashes
- Transaction volume: “Unit-of-work transactions are very important due to workflow issues with the current system”



# Architectural Characteristics

**Reliability:** The lack of reliability of the old monolithic system is having a very negative impact on customer satisfaction.

**Scalability:** As the customer base grows and the volume of tickets increases, the system must be able to scale to ensure tickets are not lost.

**Elasticity:** The system must be capable of dynamically scaling resources up and down based on real-time demand to ensure tickets are not lost.

**Ease of Upgrades:** Whenever a change was made with the monolith databases, it was taking too long and something else usually broke.


**Fault Tolerance:** Needs to be able to handle faults to ensure customer satisfaction.



# Coupling - Disintegrators

## Disintegrators Considered:

- **Service scope and function:** Unrelated services were combined together in the monolithic database.
- **Scalability and Throughput:** Some parts of the functionality like ticket creation might need to scale much more.
- **Extensibility:** It is hard to add additional functionality to the current monolithic architecture.



# Coupling - Integrators

Integrators: APIs and Messaging Queues

- **Workflow and choreography:** Some services need to talk to each other.
- **Database relationships:** The databases for some services are shared.
- **Database Transactions:** An ACID transaction is required between services some services.

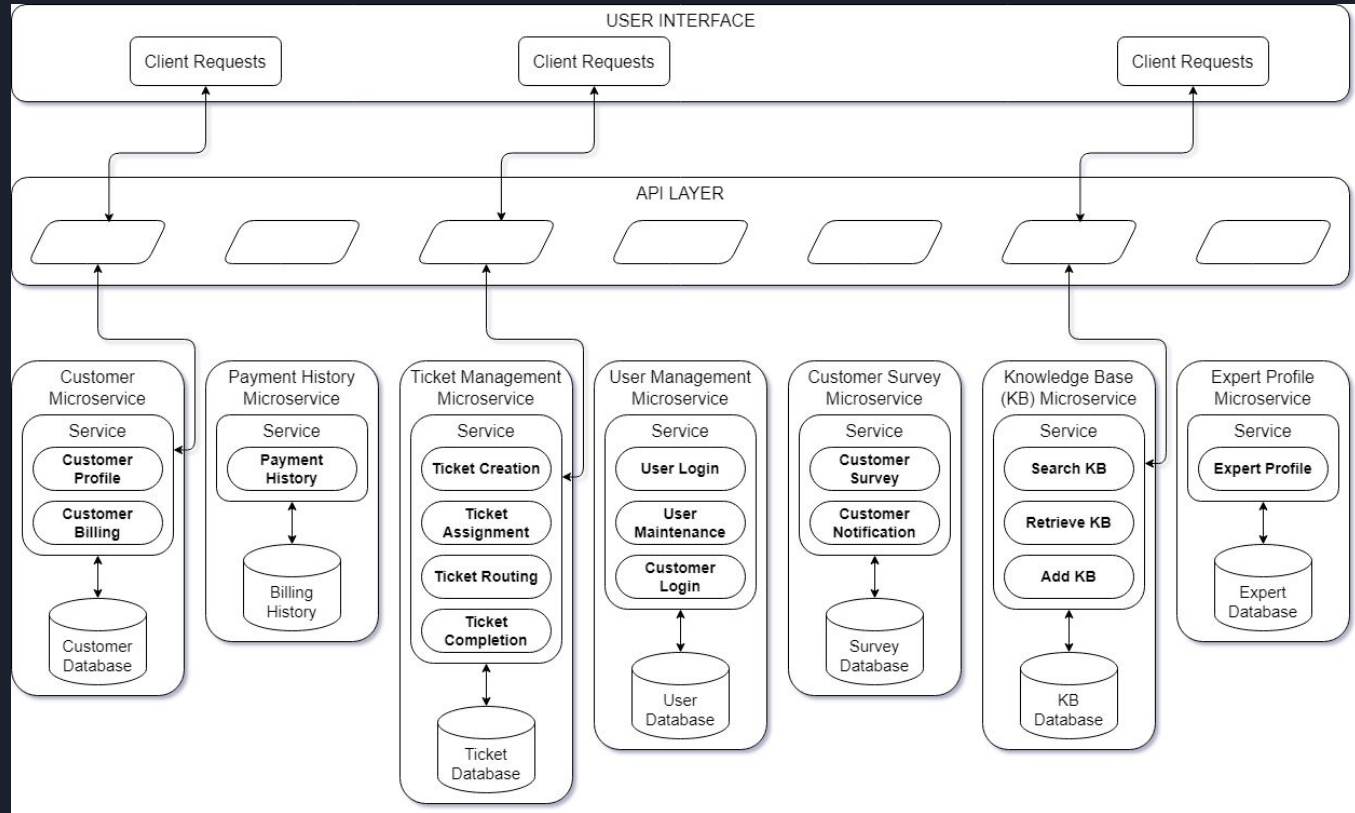


# Service Contracts

- Recommendation: Loose Contracts
  - Emphasized for the trouble ticket system's dynamic nature and need for adaptability.
  - Facilitates agile development, easing the introduction of changes without immediate disruptions.
- Consideration for Loose Contracts:
  - Flexibility requires careful versioning and communication.
  - Importance of minimizing impact on other services during changes.
- Effective Management with Loose Contracts:
  - Crucial role of comprehensive documentation.
  - Communication protocols are essential for successful implementation.
- Benefits of Loose Contracts:
  - System becomes more responsive to changes.
  - Equips Best Appliances to efficiently handle evolving demands in the trouble ticket system.



# Architecture Diagram





# Deployment Strategy - Blue-Green

## Definition:

- Strategy involves running two identical production environments: Blue (live) and Green (idle replica).
- Traffic is routed to either Blue or Green, reducing downtime and risk.

## Justification for Best Appliances:

- Zero-Downtime Deployments:
  - Essential for high availability.
- Immediate Rollback:
  - Ensures continuous service in case of post-deployment issues.
- Testing in Production:
  - Enables final testing with production settings without affecting the live environment.

## Justification for Best Appliances (Cont'd):

- Gradual Cutover:
  - Allows monitoring of new versions performance before complete switch.
- Maintenance Windows:
  - Enables maintenance or upgrades on idle environment without impacting live services.
- Performance Benchmarking:
  - Facilitates comparison to ensure the new version performs as expected.



# Deployment Strategy - Environments

## Development (DEV) Environment:

- Focus on creating and refining Blue-Green Deployment scripts.
- Implement and test data synchronization strategies.
- Develop and test automation tools within the CI/CD pipeline.

## Quality Assurance (QA) Environment

- Conduct thorough testing using the Blue-Green Deployment model.
- Evaluate data synchronization efficiency.
- Assess effectiveness of automation tools and scripts.

## Production (PROD) Environment:

- Implement Blue-Green Deployment for zero-downtime updates.
- Establish monitoring systems for issue identification.
- Continuously evaluate performance benchmarking.
- Execute gradual cutover strategies for monitoring new version performance.
- Leverage maintenance windows for seamless updates without affecting live services.



# Fitness Functions

**Cyclomatic Complexity** - Use in microservices to maintain code clarity. Set a complexity limit and employ static analysis tools to monitor compliance.

**Automatic Test Recorders** - Integrate into CI/CD pipeline to record test outcomes. Use data to ensure code stability and low defect rates in updates.

**Monitor Evaluator** - Employ monitoring tools for evaluating deployment success and resource use. Set baseline metrics and alert for anomalies.

**Dependency Assessment** - Regularly review service dependencies for loose coupling. Utilize visualization tools for identifying and addressing tightly-coupled components.



# Conclusion

- We have outlined a scalable, reliable microservices architecture to address Best Appliances' current system challenges.
- Our proposed architecture promotes ease of updates, high availability, and improved system monitoring.
- Implementing this new architecture is expected to significantly enhance customer service and operational efficiency.
- Next steps include detailed planning, phased implementation, and continuous evaluation using defined fitness functions.