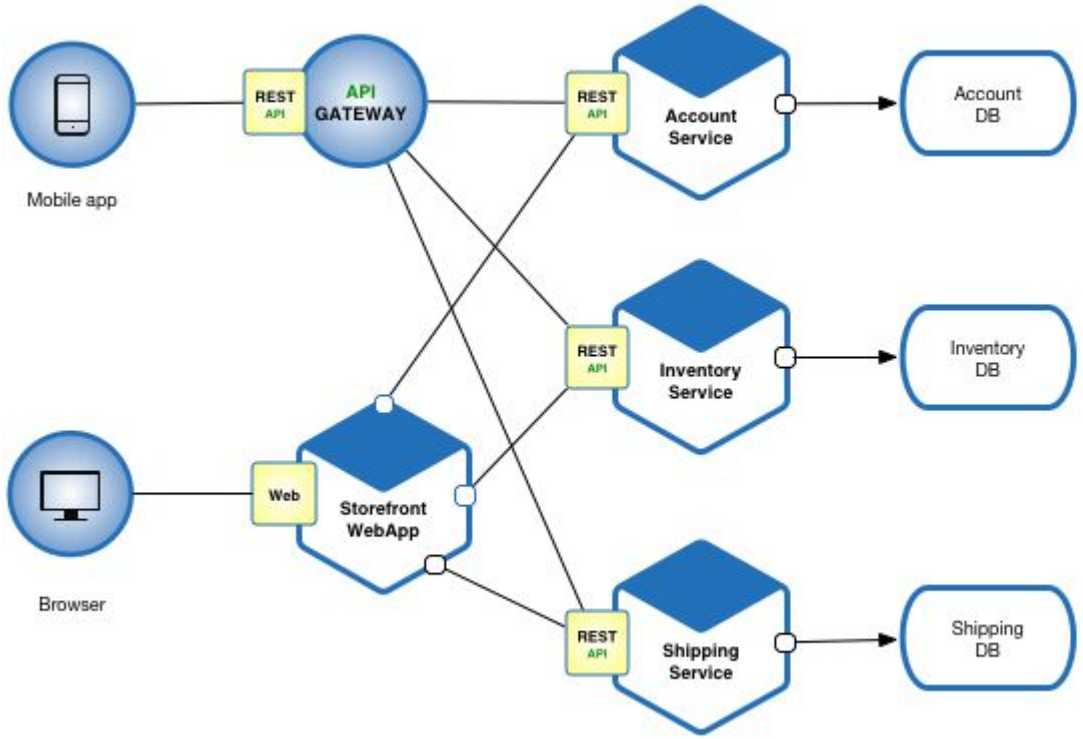
Microservices

For Software Design and Architecture (SWE 4601)

Microservices

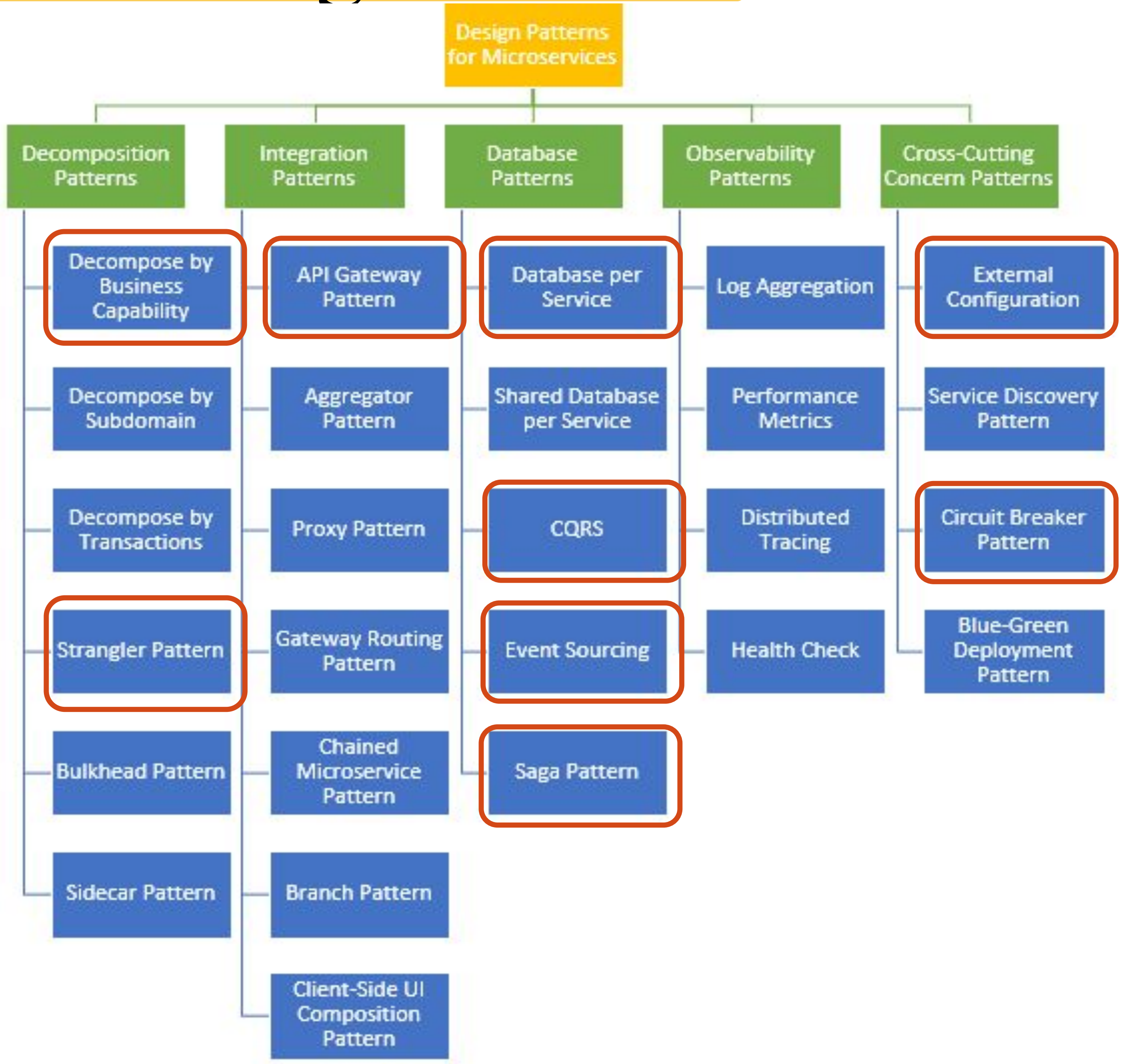
- Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities
  - Owned by a small team
- The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.



Microservice Design Patterns

- Software Design Patterns are reusable solutions to the commonly occurring problem in software Design.
- It helps us share common vocabulary and develop effective solution (i.e., Microservices)

Microservice Design Patterns



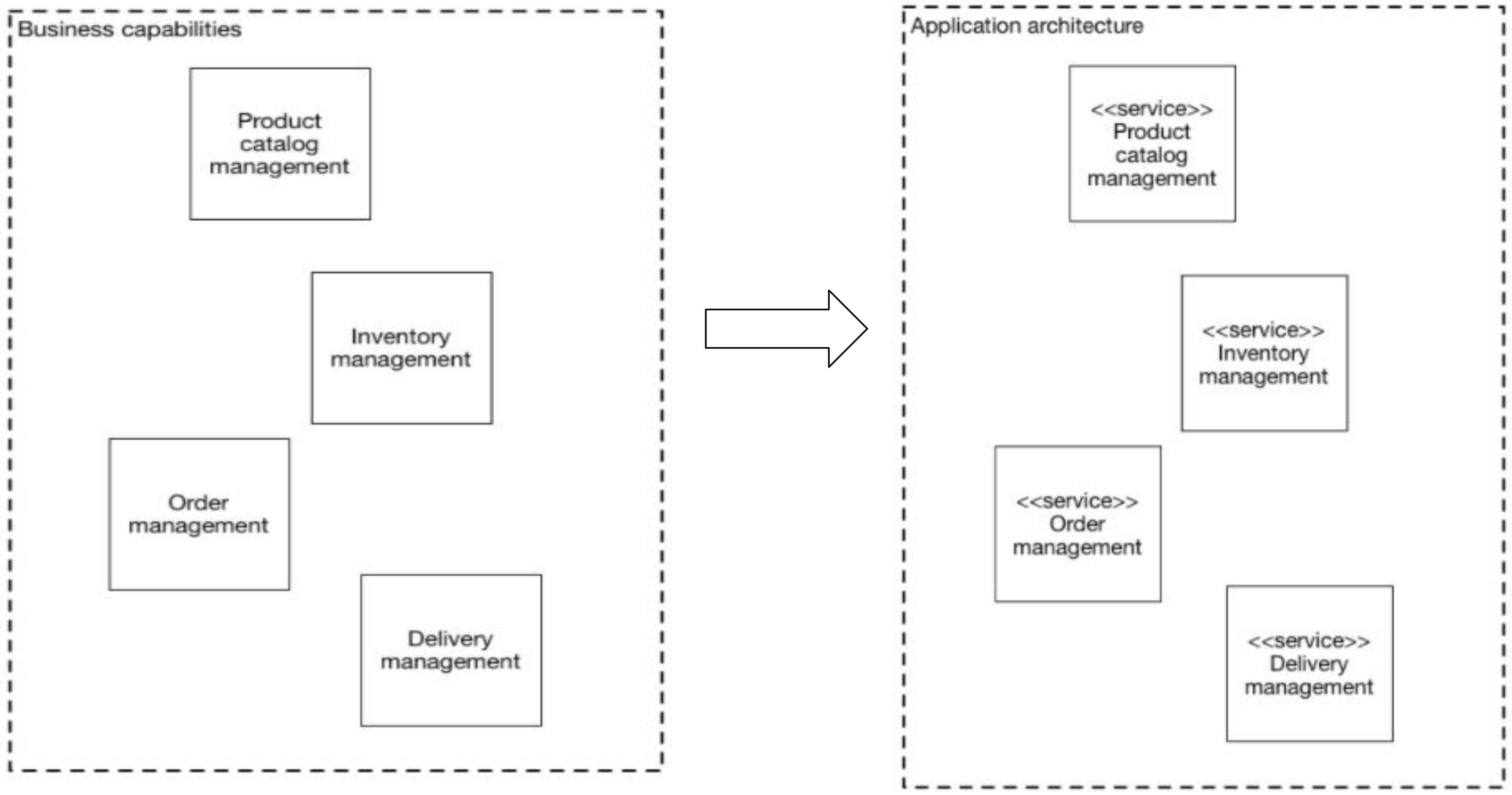
https://medium.com/@madhukaudantha/microservice-architecture-and-design-patterns-for-microservices-e0e5013fd58a

Decompose by Business Capability (Decomposition)

- Problem
  - How to decompose an application into services?
- Solution
  - Define services corresponding to business capabilities.
  - A business capability is a concept from business architecture modeling. It is something that a business does in order to generate value.
  - A business capability often corresponds to a business object, e.g.
    - *Order Management* is responsible for orders
    - *Customer Management* is responsible for customers
- Forces
  - The architecture must be stable
  - Services must be cohesive. A service should implement a small set of strongly related functions.
  - Services must conform to the Common Closure Principle - things that change together should be packaged together - to ensure that each change affect only one service
  - Services must be loosely coupled - each service as an API that encapsulates its implementation. The implementation can be changed without affecting clients
  - A service should be testable
  - Each service be small enough to be developed by a team of 6-10 people

Decompose by Business Capability (Decomposition)

Example: business capabilities of an online store


Business capabilities
Product catalog management
Inventory management
Order management
Delivery management
Application architecture
<<service>> Product catalog management
<<service>> Inventory management
<<service>> Order management
<<service>> Delivery management

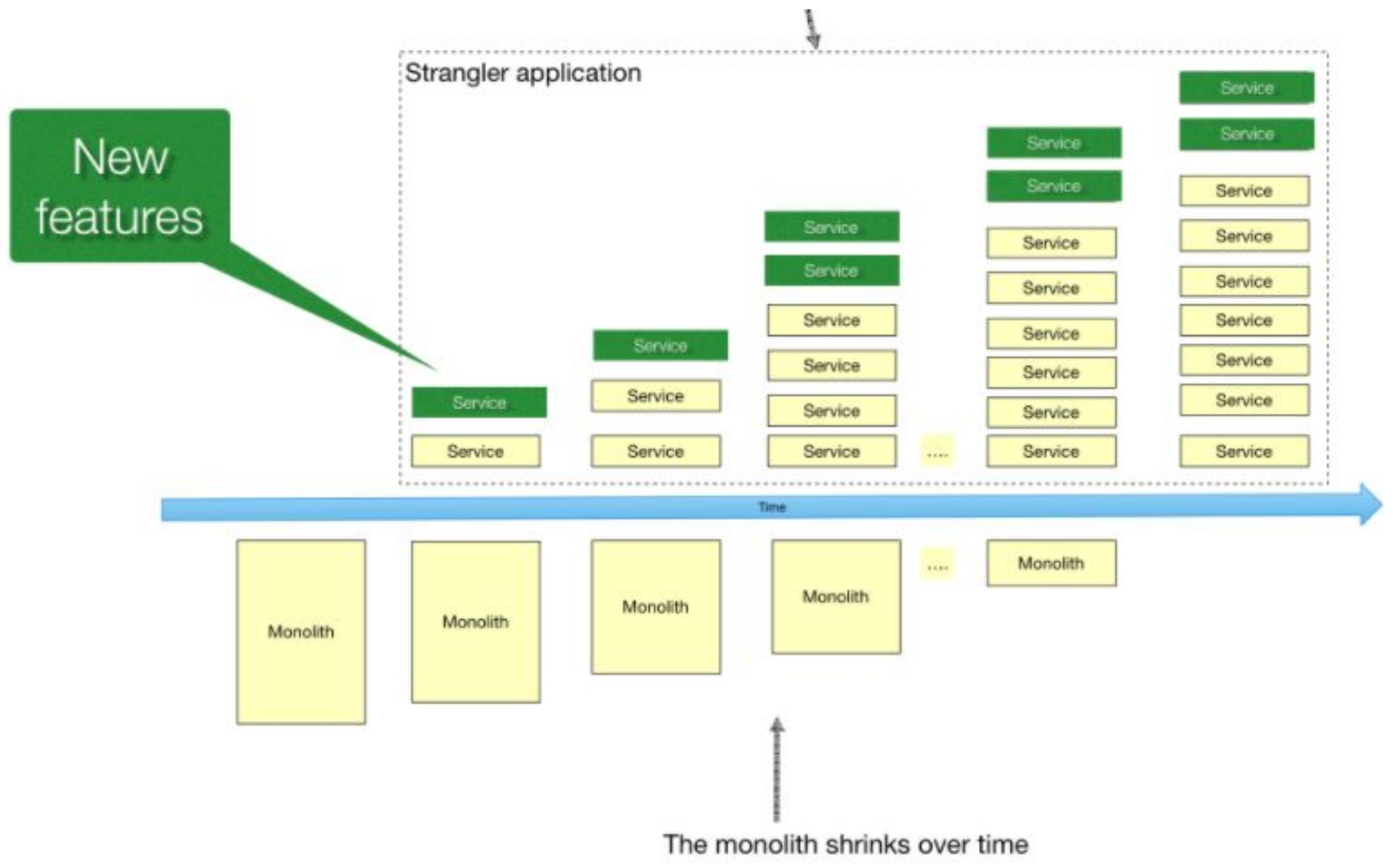
Decompose by Business Capability (Decomposition)

- Resulting Context This pattern has the following benefits:
  - Stable architecture since the business capabilities are relatively stable
  - Development teams are cross-functional, autonomous, and organized around delivering business value rather than technical features
  - Services are cohesive and loosely coupled
- Issues
  - **How to identify business capabilities?**
  - Requires understanding of the business.
  - An organization’s business capabilities are identified by analyzing the organization’s purpose, structure, business processes, and areas of expertise.
  - Bounded contexts are best identified using an iterative process.
- Related patterns
- The Decompose by SubDomain pattern is an alternative pattern

Strangler Pattern (Decomposition)

- Problem
  - How do you migrate a legacy monolithic application to a microservice architecture?
- Solution
  - Modernize an application by incrementally developing a new (strangler) application around the legacy application.
  - Decomposition done by 3 states
    - Transform
    - Co-exist
    - Eliminate

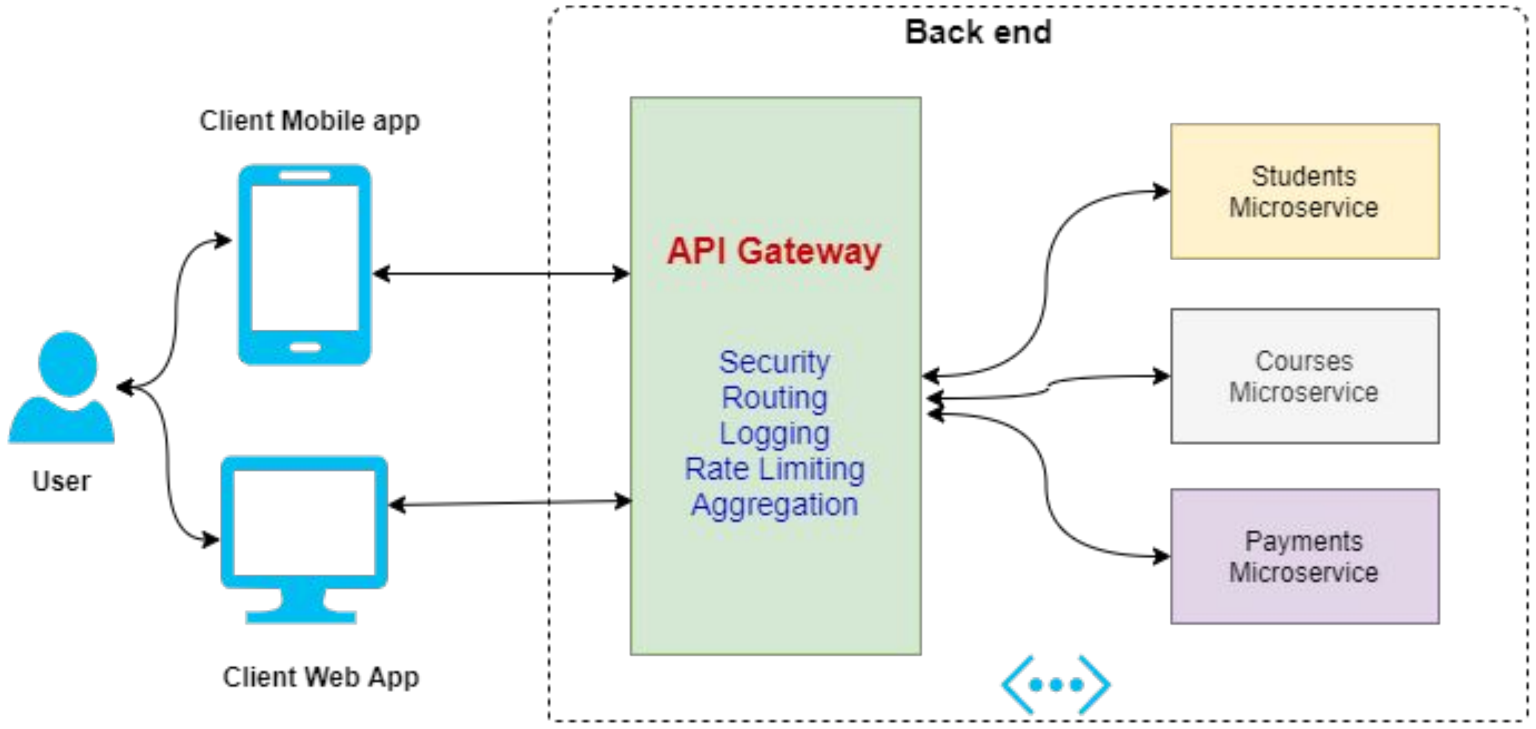
Strangler Pattern (Decomposition)



API Gateway Pattern (Integration)

- Context
  - imagine you are building an online store. you are implementing the product details page. It will be accessed mobile, desktop, web, 3rd party application
  - Since the online store uses the Microservice architecture pattern the product details data is spread over multiple services. For example,
    - Product Info Service - basic information about the product such as title, author
    - Pricing Service - product price
    - Order service - purchase history for product
    - Inventory service - product availability
    - Review service - customer reviews …
  - Consequently, the code that displays the product details needs to fetch information from all of these services.
- Problem
  - How do the clients of a Microservices-based application access the individual services?

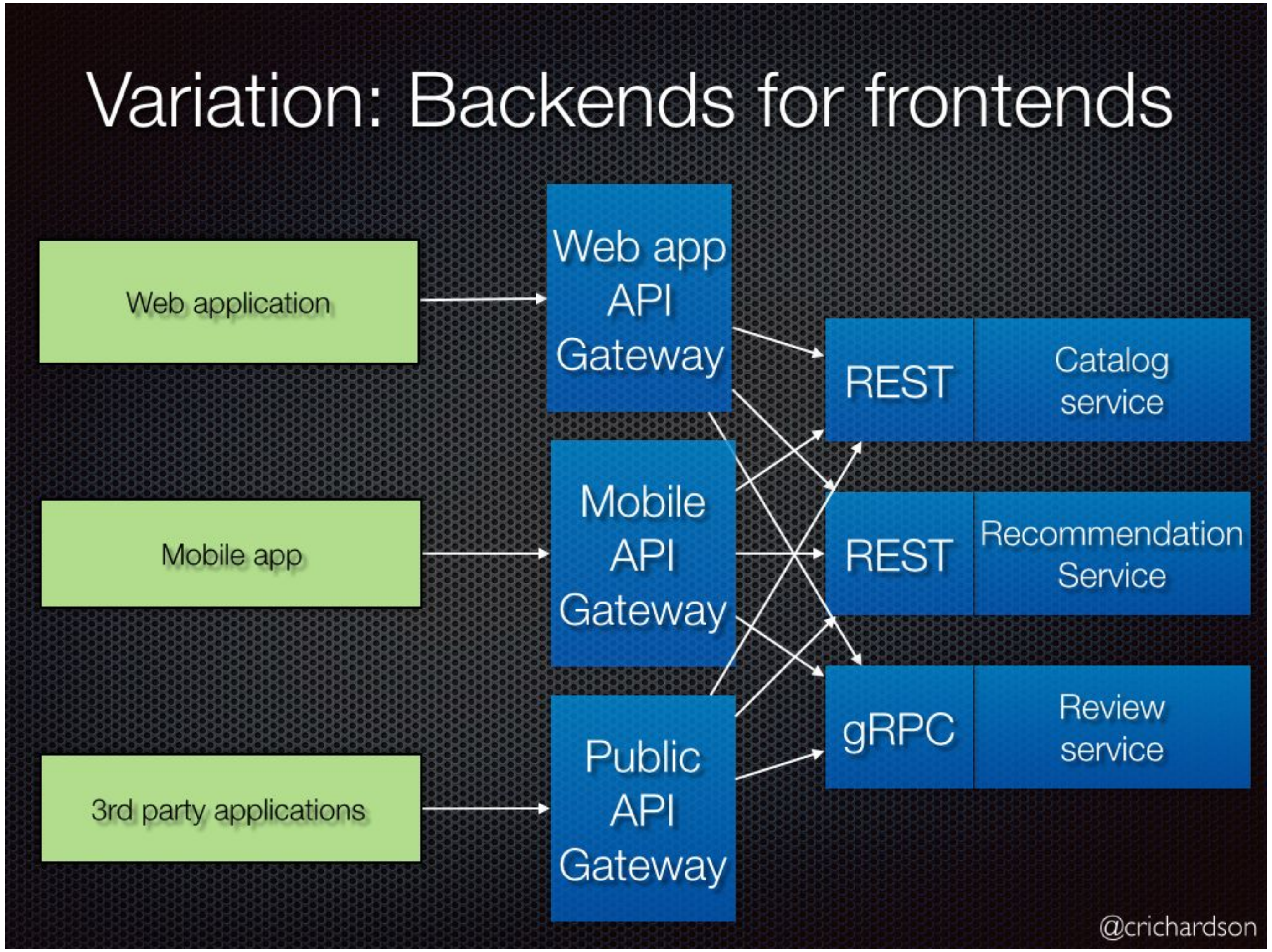
API Gateway Pattern (Integration)



API Gateway Pattern (Integration)

- Solution
  - Implement an API gateway that is the single entry point for all clients.
  - The API gateway handles requests in one of two ways. Some requests are simply proxied/routed to the appropriate service. It handles other requests by fanning out to multiple services.
  - The API gateway might also implement security, e.g. verify that the client is authorized to perform the request
- Related patterns
  - The API gateway must use either the **Client-side Discovery pattern** or **Server-side Discovery** pattern to route requests to available service instances.
  - The API Gateway may authenticate the user and pass an **Access Token** containing information about the user to the services
  - An API Gateway will use a **Circuit Breaker** to invoke services

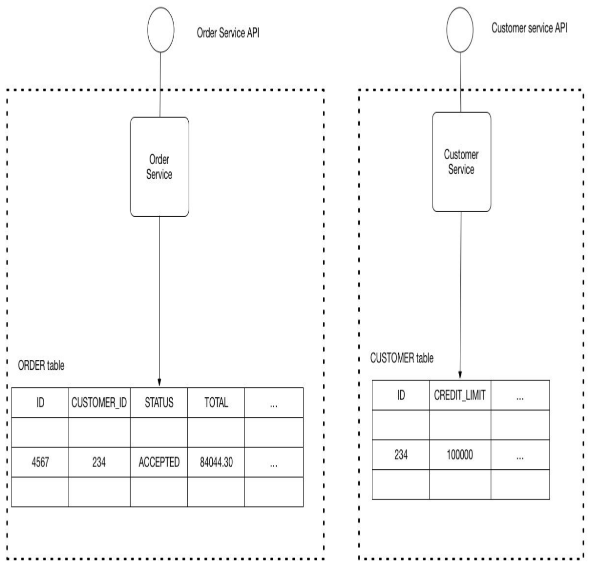
API Gateway Pattern (Integration)



Database per Service Pattern (Database)

- Problem
  - What’s the database architecture in a microservices application?
- Forces
  - Services must be loosely coupled so that they can be developed, deployed and scaled independently
  - Some queries must join data that is owned by multiple services. For example, finding customers in a particular region and their recent orders requires a join between customers and orders.
  - Databases must sometimes be replicated in order to scale
- Solution
  - Keep each microservice’s persistent data private to that service and accessible only via its API. A service’s transactions only involve its database.

Database per Service Pattern (Database)

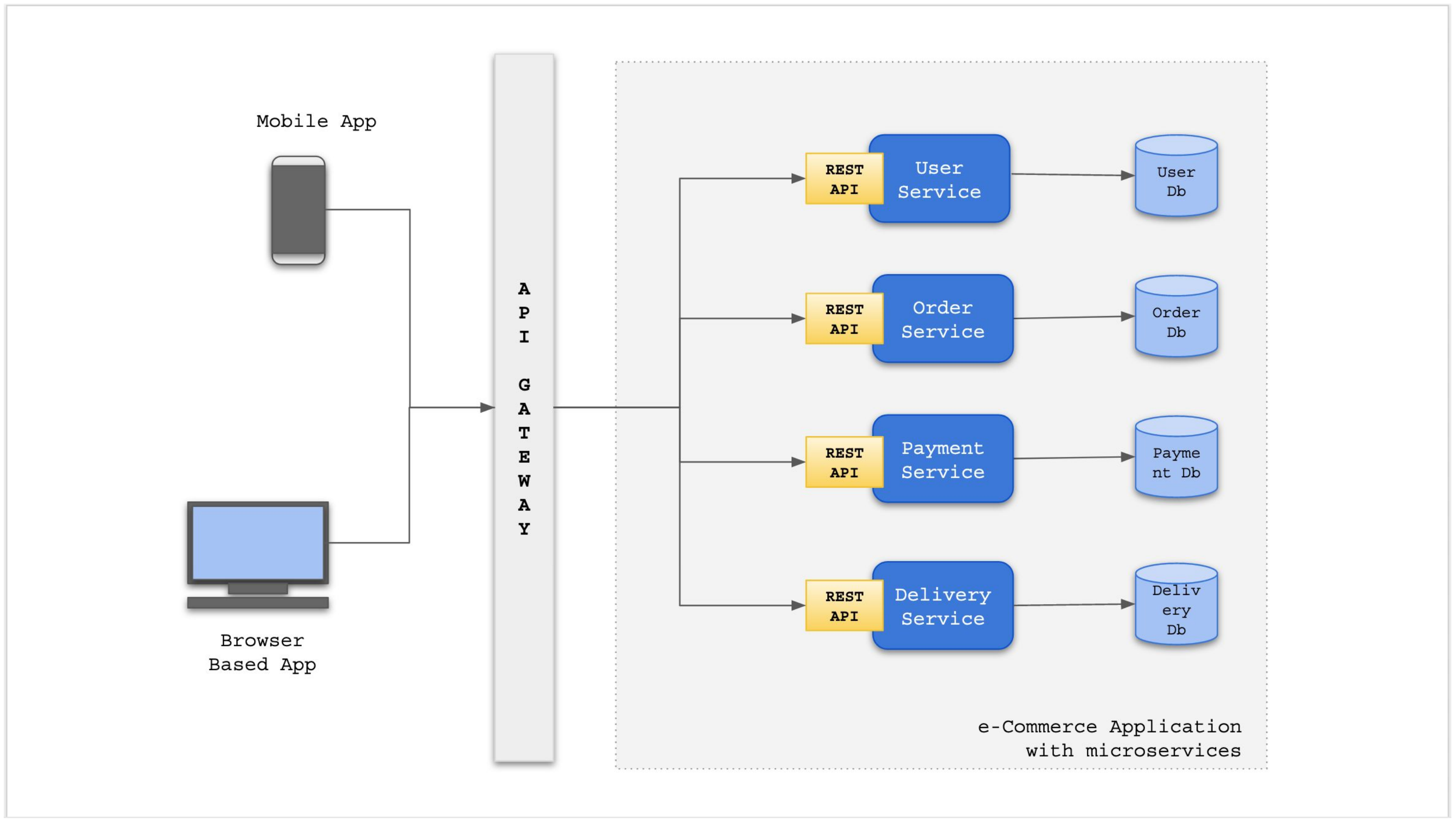


ORDER table

ID	CUSTOMER_ID	STATUS	TOTAL	...
4567	234	ACCEPTED	84044.30	...

CUSTOMER table

ID	CREDIT_LIMIT	...
234	100000	...



Database per Service Pattern (Database)

- Resulting context
  - Using a database per service has the following benefits:
  - Helps ensure that the services are loosely coupled. Changes to one service's database does not impact any other services.
  - Each service can use the type of database that is best suited to its needs. For example, a service that does text searches could use ElasticSearch. A service that manipulates a social graph could use Neo4j.
- Related patterns
  - **Saga** pattern is a useful way to implement eventually consistent transactions
  - The API Composition and **Command Query Responsibility Segregation (CQRS)** pattern are useful ways to implement queries

CQRS Pattern (Database)

- Command Query Responsibility Segregation Pattern
- Context
  - You have applied the Microservices architecture pattern and the Database per service pattern.
  - As a result, it is no longer straightforward to implement queries that join data from multiple services.
- Problem
  - How to implement a query that retrieves data from multiple services in a microservice architecture?
- Solution
  - Define a view database, which is a read-only replica that is designed to support that query. It indicates the reporting database.

References

- https://microservices.io/
- https://towardsdatascience.com/microservice-architecture-and-its-10-most-important-design-patterns-824952d7fa41
- https://medium.com/@madhukaudantha/microservice-architecture-and-design-patterns-for-microservices-e0e5013fd58a