POLITECNICO
MILANO 1863

Customers Line-up

STAY SAFE

The problem

- Covid-19
- Limited Capacity of Supermarkets
- Gatherings of People in Queues

Why Clup?

- Easy-to-use Product

- Efficient Queueing System

- Minimize Spread of Virus

The Goals

Line-Up

Booking

Tickets

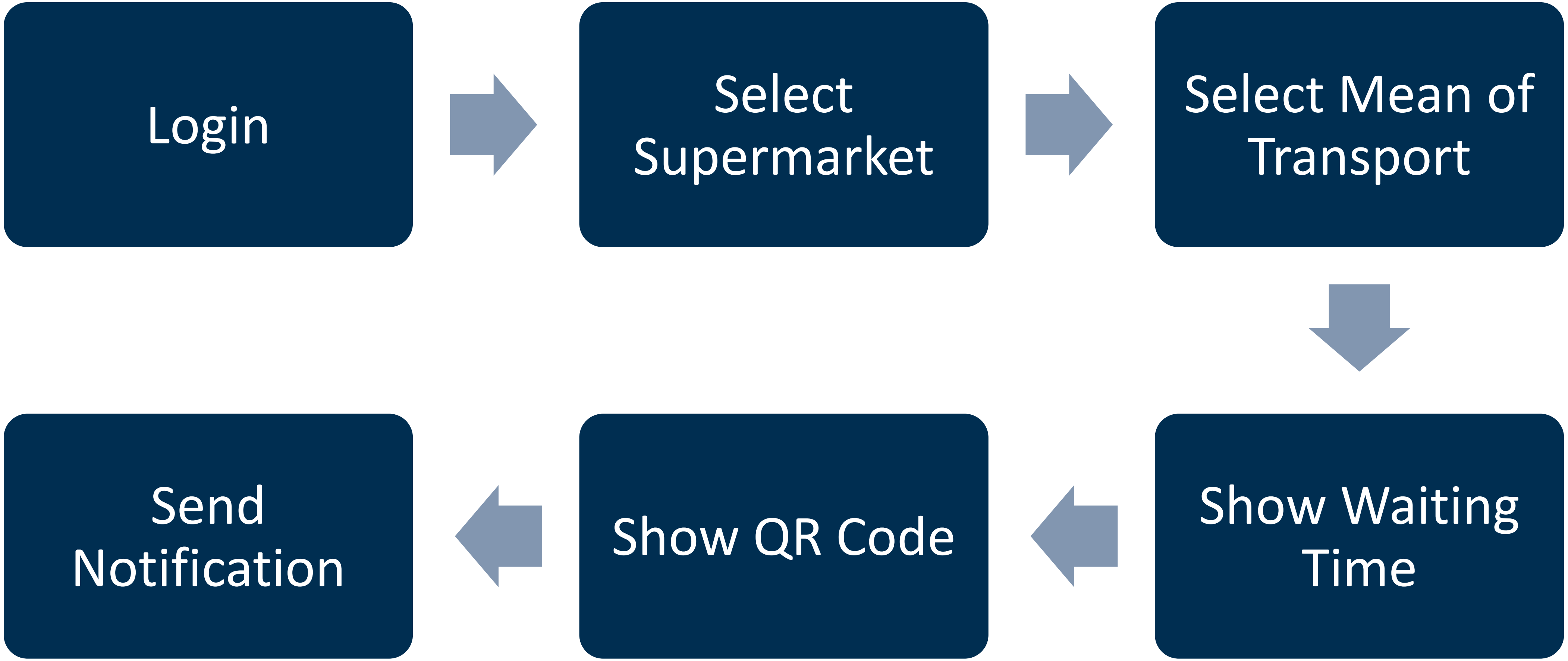
Monitor Affluence

Our Assumptions

- Only a **Small Percentage** of Customers is **No-Tech**

- Every Kind of Customer must enter Stores with **Predefined Modalities:**
  - Customer -> **Line-Up** / **Booking**
  - NoTech -> **Ticket**

Max wants to Line-Up



Login

Select Supermarket

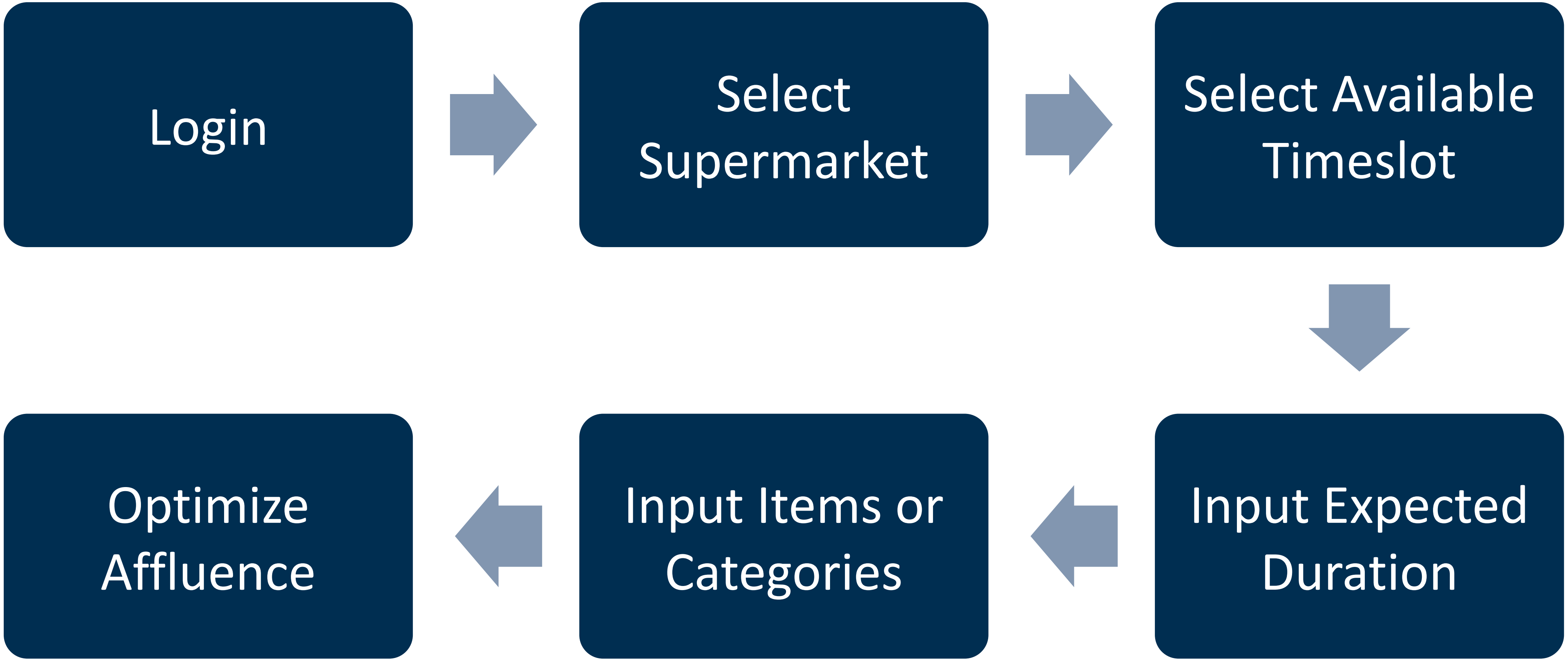
Select Mean of Transport

Show Waiting Time

Show QR Code

Send Notification

Hannah wants to Book a Visit



Login

Select Supermarket

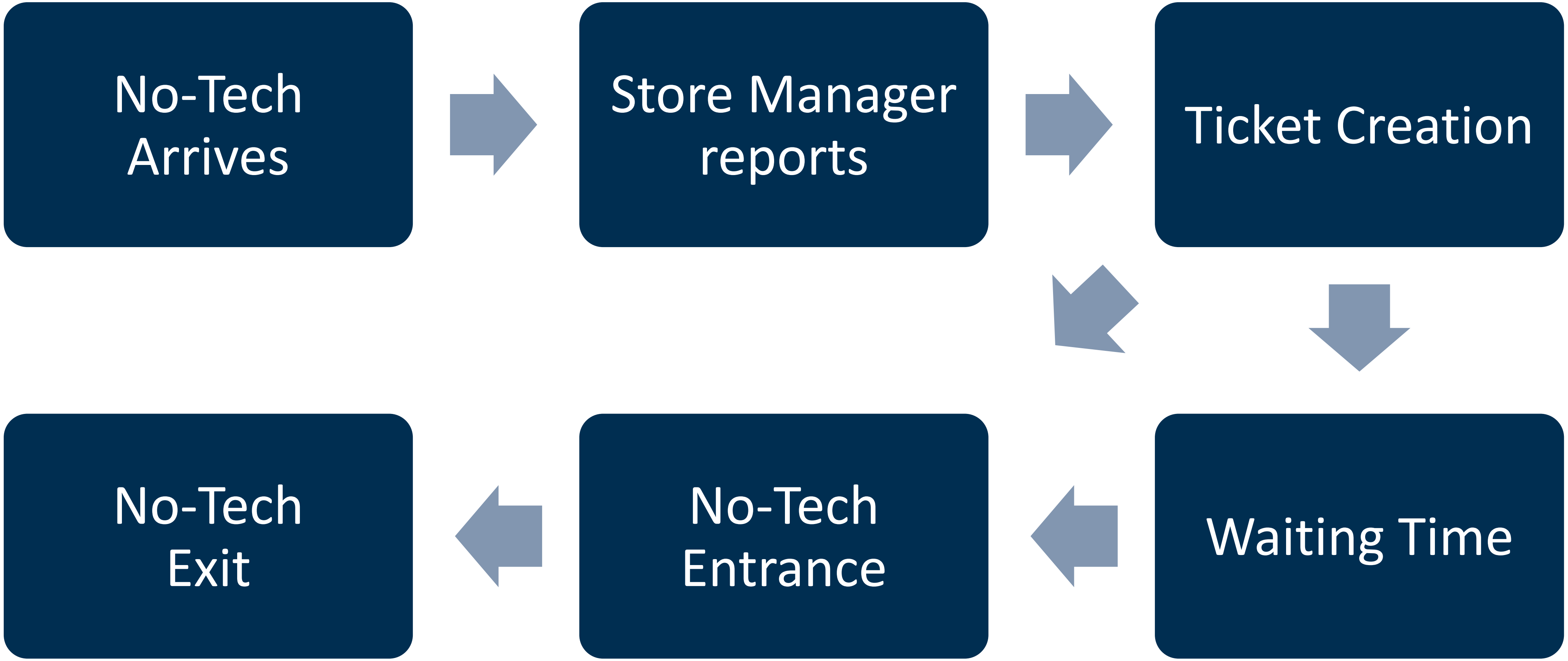
Select Available Timeslot

Input Expected Duration

Input Items or Categories

Optimize Affluence

Franck wants to enter with a Ticket



No-Tech Arrives

Store Manager reports

Ticket Creation

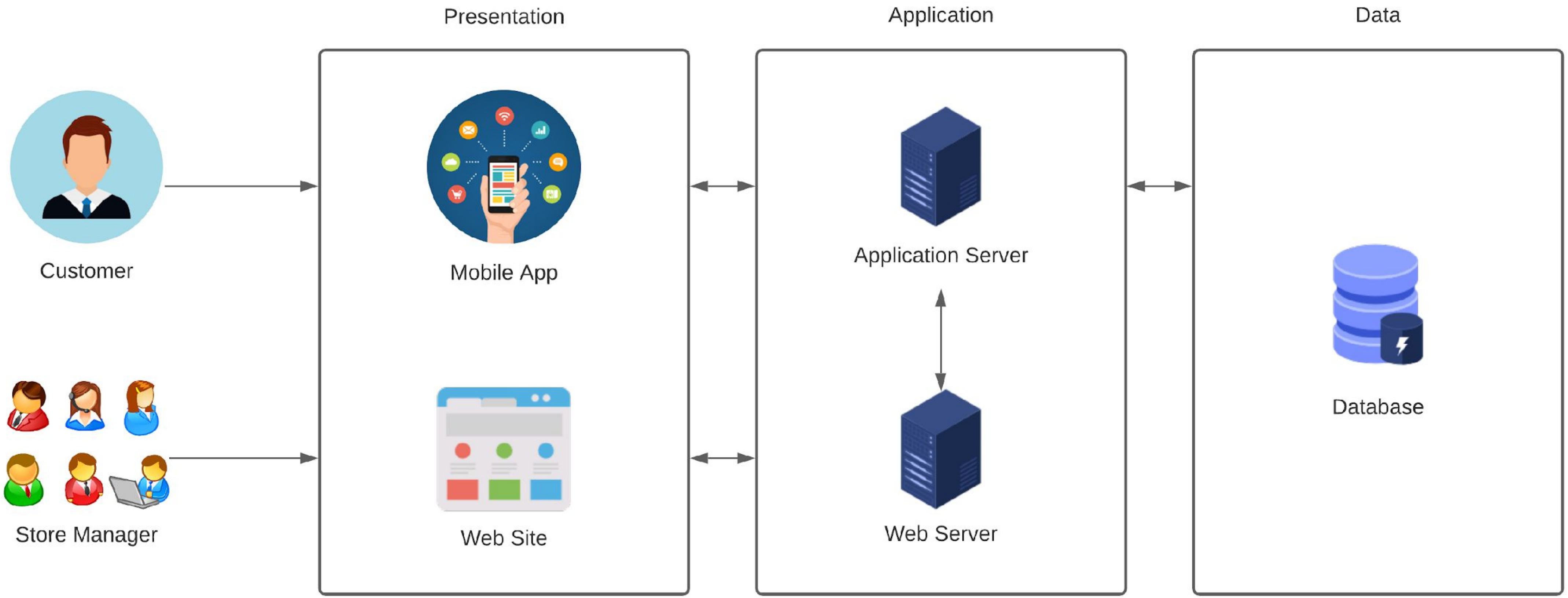
Waiting Time

No-Tech Entrance

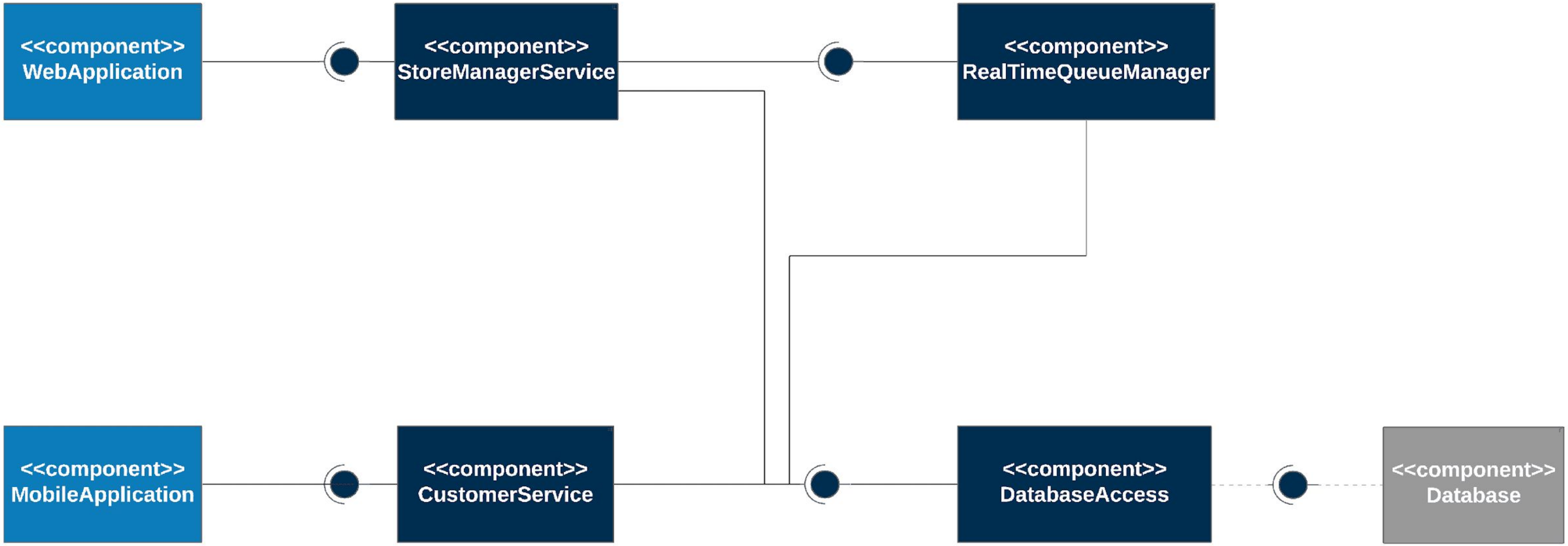
No-Tech Exit

High Level View

- **Three-Tier** Architecture

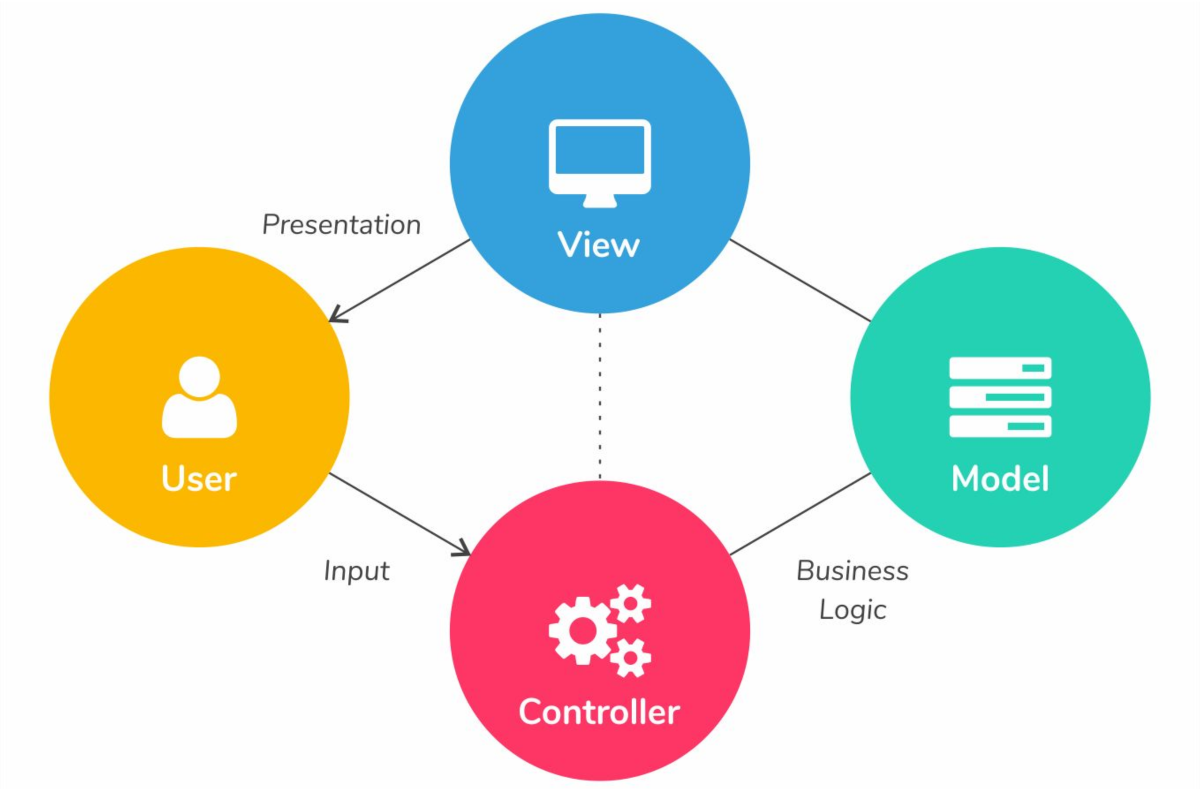


Main Components



Design Decisions & Patterns

- Model-View-Controller (MVC)



Algorithms: Waiting Time (1/6)

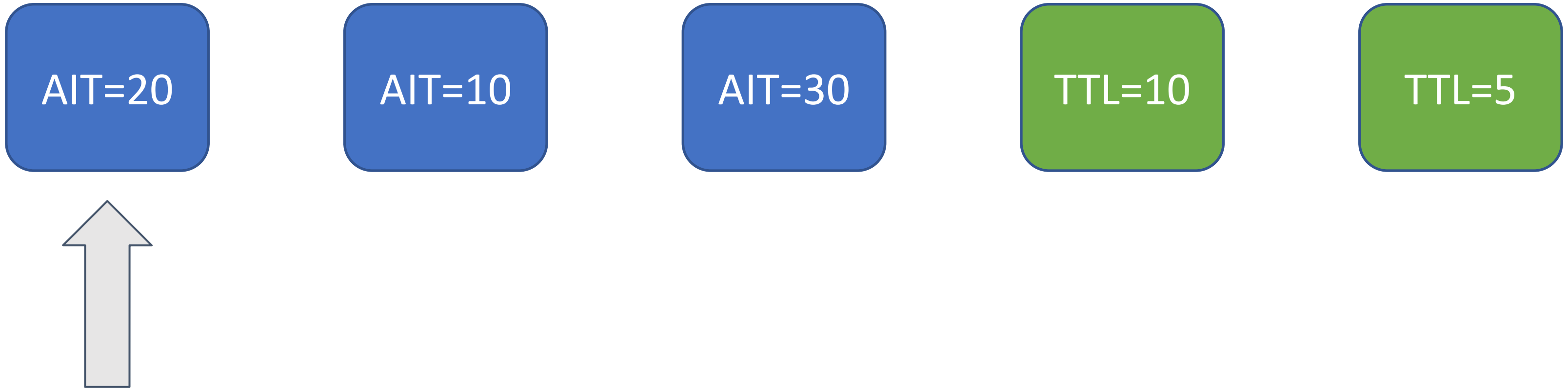
- Two Variables needed:
  1. Average Inside Time (**AIT**)
  2. Time To Leave (**TTL**)

For every Customer which precedes you, consider the TTL of Customers inside the Store, the AIT of Customers which are in Queue and their updates.

Algorithms: Waiting Time (2/6)

Partial Waiting Time

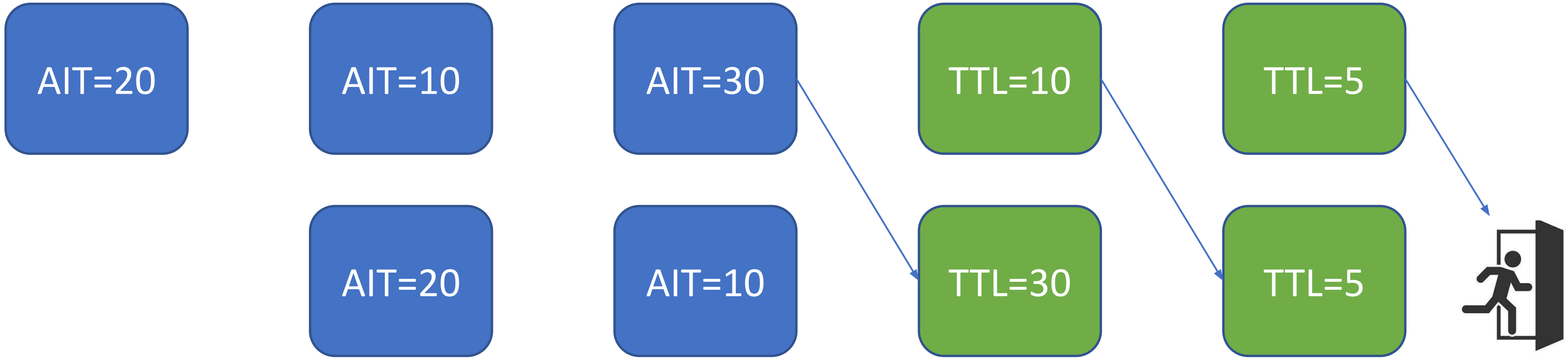
0 min



Waiting Time?

Algorithms: Waiting Time (3/6)

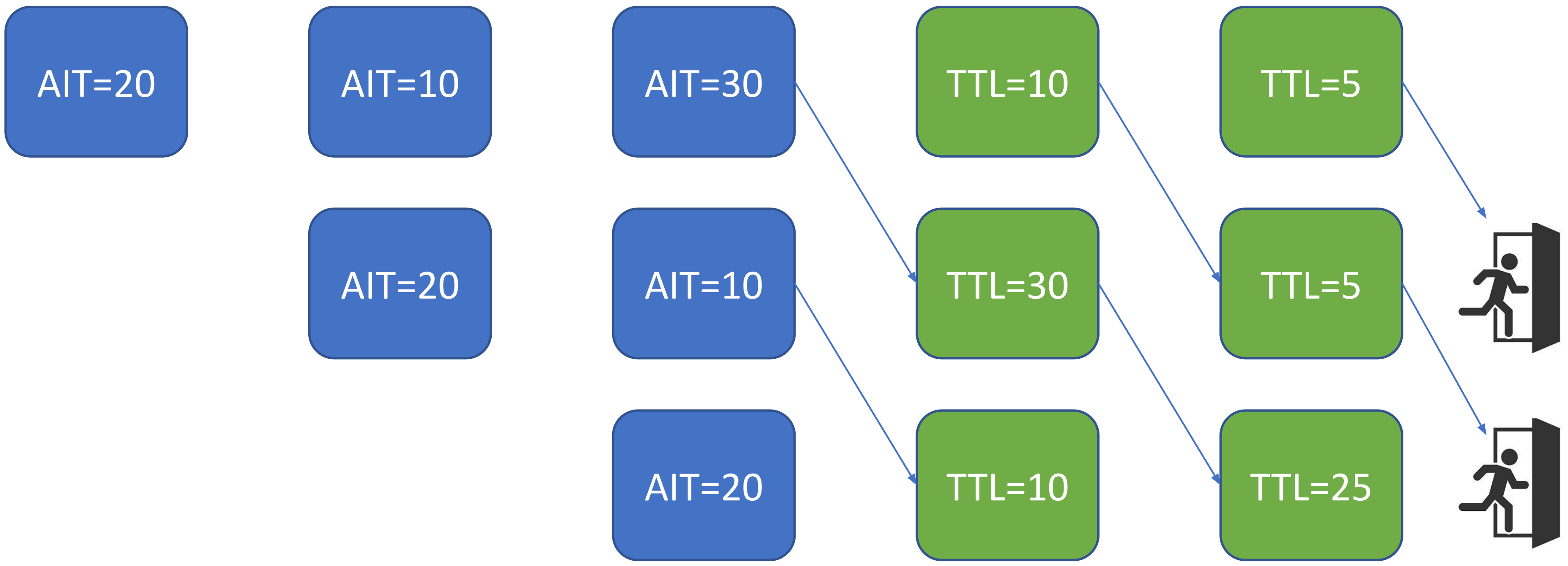
Partial Waiting Time



0 min

5 min

Algorithms: Waiting Time (4/6)



Partial Waiting Time

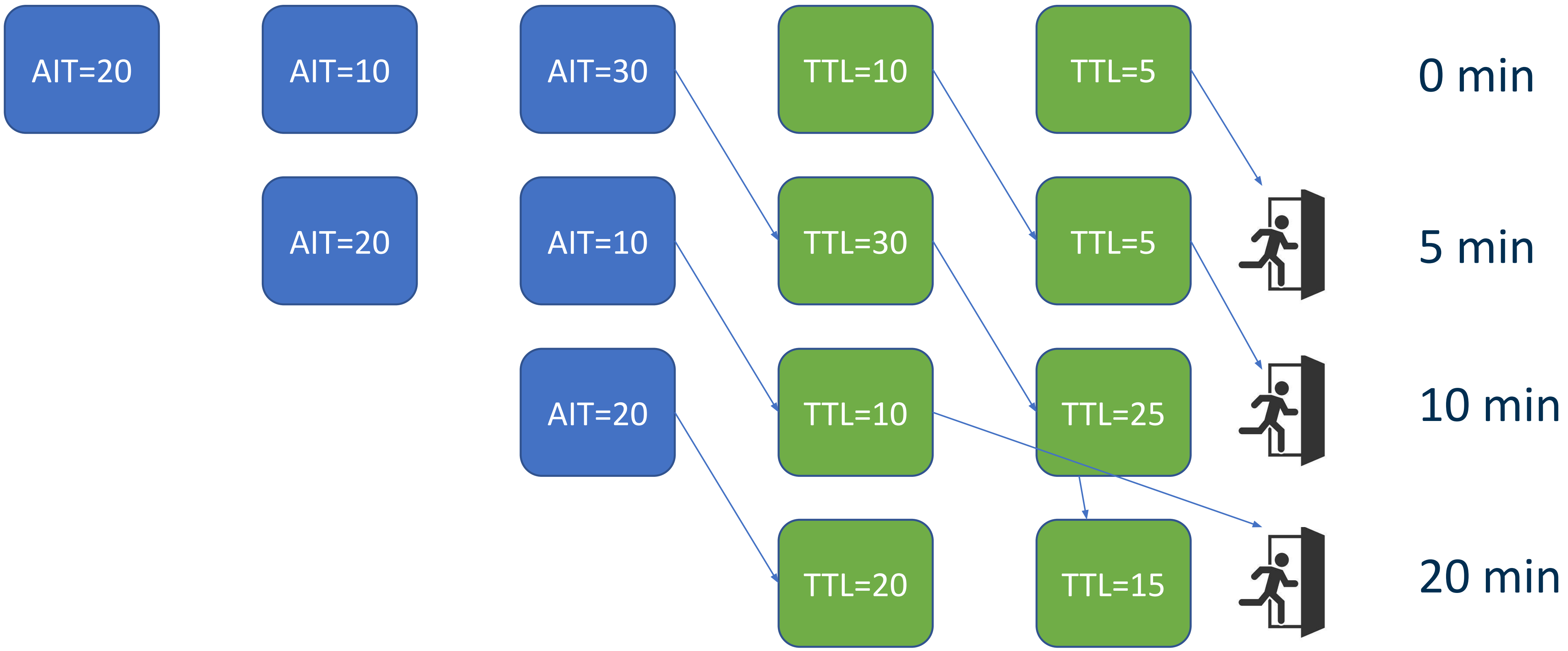
0 min

5 min

10 min

Algorithms: Waiting Time (5/6)

Partial Waiting Time



AIT=20

AIT=10

AIT=20

AIT=30

AIT=10

AIT=20

TTL=10

TTL=30

TTL=10

TTL=20

TTL=5

TTL=5

TTL=25

TTL=15

0 min

5 min

10 min

20 min

Algorithms: Implemented Waiting Time (6/6)

- Simplified Version:

Waiting Time = (PeopleInQueue - (Capacity - PeopleInside)) * AWF

- **Pros**

  - Reduced Complexity
  - Less Data required

- **Cons**

  - Less Precise

Framework & Technologies

React Native

Presentation
Tier

Java EE™

Application
Tier

MySQL®

Data
Tier

Implementation Plan

POLITECNICO MILANO 1863

- **Bottom-Up** Approach

- Pros:
  - **Simple**
  - **Efficient**
  - **Better tracking** in case of Bugs

- Cons:
  - **Cannot release immediately an “early version”**

Integration & Testing

- **Bottom-Up** Approach
- **Unit** Testing: JUnit for Application Server
- **Integration** Testing: tested RESTFul APIs and interaction with Business Logic, Entities and Database
- **System** Testing: performed manually and by another team

Thank You for Your Attention!

DEMO
(code)

Algorithms: Booking Optimization (Backup 1/2)

- **Upper Bound Capacity** established by the DPCM

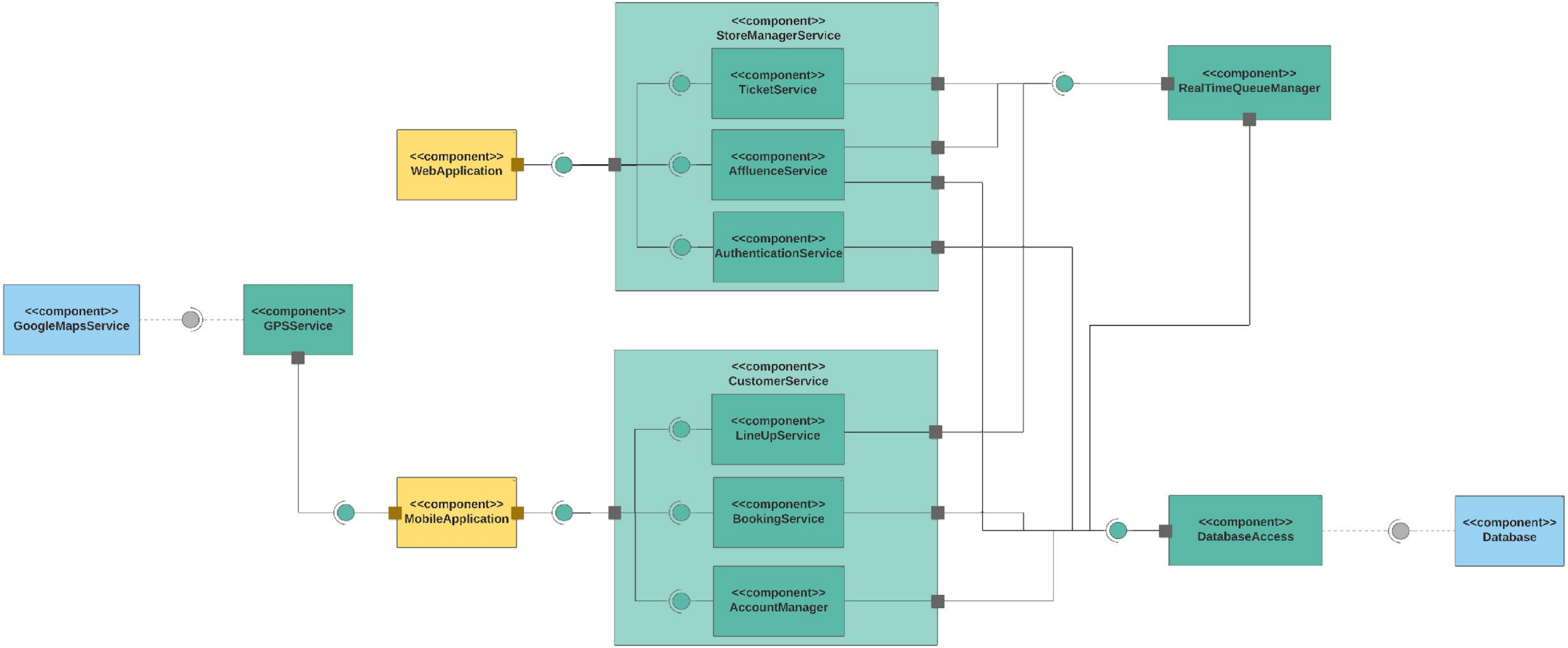
- For every Customer, with respect to the Items that he has indicated we compute the Probability for him to be in a specific Department at a certain time *t.*

- For each time *t*, compute the **Average Affluence** of every Department

- If no Department is in **Risk of Overcrowding**, then increase the Capacity as:

95% + (5% * (BookedPeople / Capacity))

Component Diagram (Backup 2/2)


<<component>>
StoreManagerService
<<component>>
TicketService
<<component>>
AffluenceService
<<component>>
AuthenticationService
<<component>>
RealTimeQueueManager
<<component>>
WebApplication
<<component>>
GoogleMapsService
<<component>>
GPSService
<<component>>
CustomerService
<<component>>
LineUpService
<<component>>
BookingService
<<component>>
AccountManager
<<component>>
MobileApplication
<<component>>
DatabaseAccess
<<component>>
Database