

# **Zetra Hotel Document**

# USER STORY

Hotel Zetra adalah hotel yang sudah berdiri selama 5 tahun. Hotel ini sudah berkembang pesat sehingga ingin melakukan ekspansi. Berikut adalah beberapa kondisi pada hotel zetra yang lama dan yang akan datang.

Hotel Zetra memiliki sebuah sistem dimana calon penghuni hotel / pelanggan harus melakukan booking dulu sebelum datang dan menikmati kamar di hotel ini. Booking dilakukan boleh dari jauh hari, atau dari hari-h (Booking dilakukan pada hari ini dan datang hari ini). Ketika melakukan booking, pelanggan akan diminta untuk memilih tipe kamar yang diinginkan, kemudian tanggal awal sampai tanggal akhir menginap. Sistem kemudian akan mencari kamar yang memiliki tipe yang diinginkan dan kamar tersebut belum ada yang menghuni dari tanggal awal booking sampai tanggal akhir booking. Kamar yang ditemukan sistem akan dipilih 1 untuk data booking pelanggan. Jika tidak ditemukan maka akan ada pesan bahwa kamar tidak ditemukan untuk tanggal yang diinginkan. Tanggal booking itu dilakukan dan pegawai yang melayani booking tersebut juga dicatat. Pelanggan akan dicatat data pribadi (sesuai KTP) dan no KTP pelanggan.

Sistem pada hotel ini, jika pelanggan ingin melakukan pesanan harus melalui front desk. Artinya belum ada platform online yang memberikan layanan pemesanan. Pegawai yang ada pada front desk akan melakukan login untuk mencatat pesanan pelanggan.

Ketika pelanggan datang untuk check-in, front desk akan mencari ada order pelanggan sesuai dengan nomor KTP yang digunakan untuk booking. Ada beberapa kondisi ketika pelanggan datang untuk melakukan check-in.

Kondisi - kondisi waktu check-in:

- Ketika check-in pelanggan datang lebih awal dari jam 12, pada kasus ini pelanggan akan dipersilahkan untuk menunggu.
- Ketika check-in pelanggan datang sesuai dengan waktu yang ditentukan ( $\geq 12$ ) dan kamar sudah dalam status siap, maka pelanggan dapat melakukan check-in.
- Ketika check-in pelanggan datang sesuai dengan waktu yang ditentukan ( $\geq 12$ ), tetapi kamar yang akan dihuni oleh pelanggan belum siap. Pada kasus ini sistem akan mencari kamar dengan tipe yang sama tetapi memiliki status siap & penghuni kamar tersebut belum

datang untuk melakukan check-in. Kemudian sistem akan menukarkan 2 kamar tersebut sehingga pelanggan yang sudah datang dapat melakukan check-in.

- Jika poin 3 di atas tidak dapat dilakukan karena tidak ada kamar lainnya yang siap dihuni maka pelanggan akan mendapatkan kompensasi berupa voucher makan di restoran hotel. Kasus ini harus dicatat dan dianggap sebagai accident.
- Jika poin 3 di atas tidak dapat dilakukan karena tidak ada kamar lagi dengan tipe yang sama yang disewa oleh pelanggan, maka pelanggan akan mendapatkan kompensasi dengan menaikkan tipe kamar menjadi tipe yang lebih tinggi. Kasus ini harus dicatat dan dianggap sebagai accident.

Hotel Zetra juga memiliki sistem dimana masing - masing property / barang yang ada pada hotel tersebut diberikan sebuah kode QR / Barcode. Sistem ini dilakukan untuk melakukan pelacakan barang yang ada. Barang - barang dalam hotel ini kebanyakan tersebar di semua kamar hotel ini. Jika ada barang yang hilang dan rusak maka pegawai akan mengubah status barang sesuai dengan kondisi yang terjadi. Setelah itu demi tetap memberikan rasa nyaman pada pelanggan, hotel ini akan mengambil barang baru pada gudang menuju kamar hotel. Proses ini disebut sirkulasi barang. Proses sirkulasi tidak hanya menukarkan barang dari kamar menuju gudang tetapi juga terjadi ketika ada barang yang dipesan hotel ini datang dan masuk ke dalam gudang. Pemesanan barang pada hotel ini disebut sebagai Purchase Order dimana pegawai hotel akan melakukan input pemesanan barang kepada supplier hotel.

Proses Purchase Order pada Hotel Zetra terjadi dengan beberapa urutan kerja: ● Pegawai Hotel Zetra melihat catalog supplier, kemudian memilih barang apa saja yang ingin dibeli.

- Pegawai membuat data Purchase Order pada aplikasi kemudian data PO tersebut akan memiliki status awal CREATED.
- Kemudian data PO tersebut akan dievaluasi oleh manajer hotel. Jika PO tersebut disetujui maka PO akan disampaikan menuju supplier. Hotel juga membayar DP / Down Payment dari PO tersebut. Status PO menjadi ACCEPTED.
- Jika barang yang dipesan sudah datang maka status PO menjadi DELIVERED dan proses sirkulasi terjadi.
- Jika PO sudah dilunasi maka status PO menjadi FINISHED.

- Jika pada poin 3 manajer tidak menyetujui PO, maka status PO menjadi REJECTED.

Hotel ini juga memiliki sistem pada restoran. Pada restoran hotel, tiap hari restoran harus menyediakan breakfast untuk semua pengunjung hotel. Setiap hari koki restoran akan menentukan menu yang ingin dibuat dengan jumlahnya. Kemudian untuk setiap menu sudah memiliki detail bahan apa saja yang harus disiapkan. Barang - barang tersebut termasuk menjadi barang - barang yang ada pada gudang. Restoran ini juga dapat

memasakan makanan untuk tamu hotel. Jika ada pesanan makanan yang masuk, entah dari pelanggan atau proses breakfast maka menu yang dibentuk akan melakukan proses pengurangan qty barang pada gudang. Mini bar pada kamar yang berisi snack dan minuman akan berubah - ubah item yang ada, proses pertukaran item makanan pada mini bar akan masuk kedalam sistem sirkulasi barang.

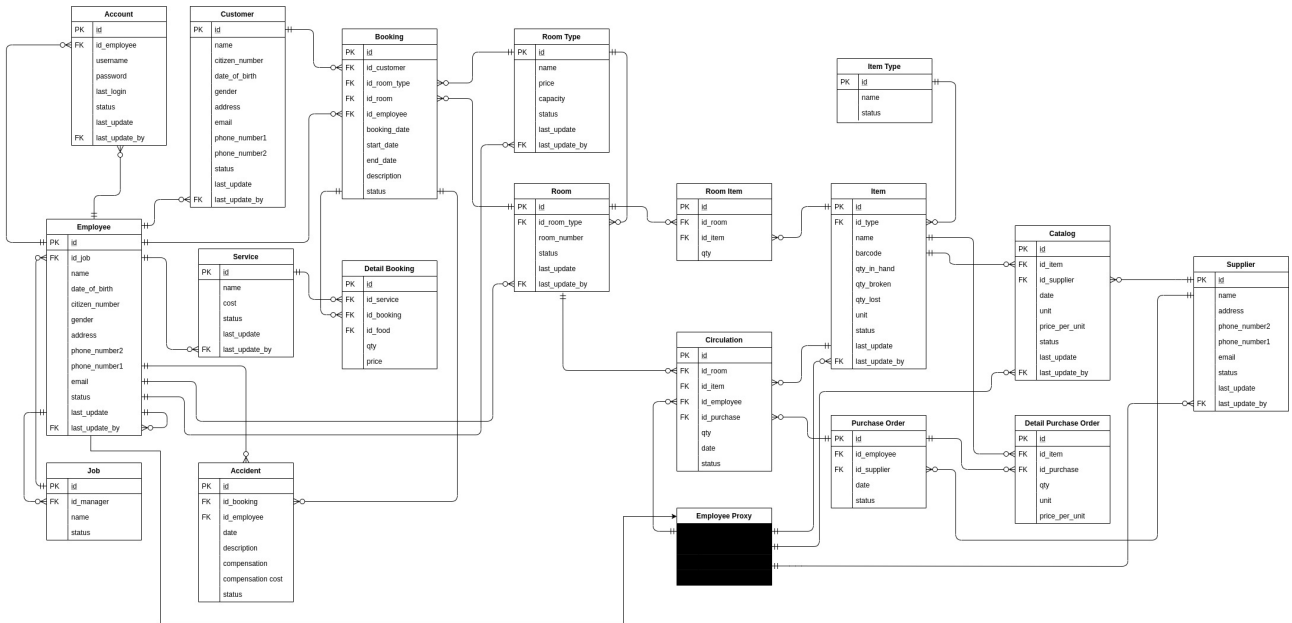
Accident management system adalah subsystem dan termasuk dalam sistem informasi besar hotel. Sistem ini mencatat semua kejadian - kejadian pada Hotel Zetra yang dapat mengurangi kenyamanan pelanggan untuk tinggal dalam hotel tersebut. Masing - masing accident memiliki category-nya sendiri. Setiap accident akan dicatat, siapa yang membuat laporan (pegawai), tanggal, deskripsi kejadian, siapa tamu yang terlibat (jika ada), category accident.

Dashboard merupakan bagian dari sistem dimana pengguna yang memiliki akun dengan tingkat manajer dapat masuk dan melihat laporan - laporan yang ada. Laporan - laporan pada sistem ini meliputi semua data yang ada pada Hotel Zetra.

# ERD HOTEL ZETRA

Untuk melihat gambar ERD lebih jelas, silahkan klik pada link berikut :

<https://ibb.co/9Gj155F>



## DEPENDENCIES

# BOOKING

## Import Libraries

```
1 from nameko.extensions import DependencyProvider
2 import mysql.connector
3 from mysql.connector import Error
4 from mysql.connector import pooling
```

Import DependencyProvider untuk menyembunyikan code yang bukan bagian dari Service Logic. Interface dari dependency ke service harus sesederhana mungkin. Dan untuk mendeklarasikan dependency ke dalam service adalah langkah yang bagus karena :

- Sudah meng-includekan service yang lain, beberapa external API, dan database juga

Dalam kasus ini, kami membuat dependency untuk membuat koneksi ke MySQLDatabase melalui 'connection pool'. Kita dapat menggambarkan DBDependencies sebagai 'connection pool'. Ketika ada request service instance yang akan dibuat, maka DBDependencies yang menginjeksikan menggunakan koneksi dari 'connection pool' tersebut.

## Connect Microservice to Database

Berikut adalah contoh mengkoneksikan MySQL Database dengan microservice kami melalui dependencies :

```
1 class Database(DependencyProvider):
2     connection_pool = None
3     def __init__(self):
4         try:
5             self.connection_pool = mysql.connector.pooling.MySQLConnection
6                 pool_name="database_pool",
7                 pool_size=5,
8                 pool_reset_session=True,
9                 host='localhost',
10                database='proyek soa 2',
```

```

11         user='root',
12         password=''
13     )
14     except Error as e :
15         print ("Error while connecting to MySQL using Connection pool
16 def get_dependency(self, worker_ctx):
17     return DatabaseWrapper(self.connection_pool.get_connection())

```

Untuk bagian **host** diisi dengan '**localhost**' karena kami menggunakan komputer pribadi sebagai virtual server untuk mengakses database. Untuk mengakses localhost, biasanya harus menggunakan alamat IP default **127.0.0.1** atau disebut juga dengan loopback address.

Untuk **database** diisi dengan '**proyek soa 2**' karena waktu membuat database kami memberi nama 'proyek soa 2'. Ini disesuaikan dengan nama database agar dapat terkoneksi dengan database yang sudah kita buat

Untuk **user = 'root'** dan **password = ''** adalah username dan password default dari mySQL Databasenya. User '**root**' dalam istilah keamanan komputer sering disebut sebagai '**superuser**'. **Superuser** merupakan tingkatan user tertinggi dimana user ini dapat melihat, mengubah, bahkan menghapus seluruh database dan menjalankan perintah apapun yang terdapat dalam MySQL.



**XAMPP** menggunakan **user = 'root'** dan **password = ''**

## Class DatabaseWrapper

Setelah itu, kami dapat membuat sebuah Class bernama **DatabaseWrapper** yang berisi fungsi-fungsi yang akan digunakan dalam service class instances.

```

booking_dependencies.py

1 class DatabaseWrapper:
2
3     def __init__(self, connection):
4         self.connection = connection

```






Setelah syntax SQL berhasil ditulis, maka cursor akan mengeksekusi SQL tersebut dan juga mengambil semua data yang diperintahkan oleh query. Setelah itu kami menyimpan data yang telah diquery ke dalam variabel '**result**'. Lalu cursor akan di close dan hasil dalam variabel 'result' akan di-return kan.

Hal yang sama juga diterapkan pada fungsi '**add\_customer**' dimana fungsi ini menerima beberapa parameter yang berfungsi untuk menerima passing data. Data tersebut akan digunakan untuk mengisi kolom-kolom yang dibutuhkan pada tabel tersebut (dalam kasus ini pada tabel **customer**).

```
val = (name, citizen_number, date_of_birth, gender, address, email, phone_number1,  
phone_number2, status, employee_id)
```

Variabel yang ada dalam **val** harus urut mengikuti susunan pada query.

 Penjelasan lebih lanjut tentang fungsi yang ada pada **booking service** dapat dilihat pada halaman [RPC Booking](#)

# ACCIDENT

## Import Libraries

```
1 from nameko.extensions import DependencyProvider
2 import mysql.connector
3 from mysql.connector import Error
4 from mysql.connector import pooling
```

Import `DependencyProvider` untuk menyembunyikan code yang bukan bagian dari `Service Logic`. Interface dari dependency ke service harus sesederhana mungkin. Dan untuk mendeklarasikan dependency ke dalam service adalah langkah yang bagus karena :

- Sudah meng-includekan service yang lain, beberapa external API, dan database juga

Dalam kasus ini, kami membuat dependency untuk membuat koneksi ke `MySQLDatabase` melalui 'connection pool'. Kita dapat menggambarkan `DBDependencies` sebagai 'connection pool'. Ketika ada request service instance yang akan dibuat, maka `DBDependencies` yang menginjeksikan menggunakan koneksi dari 'connection pool' tersebut.

## Connect Microservice to Database

Contoh mengkoneksikan `MySQL Database` dengan microservice kami melalui dependencies :



Untuk penjelasan bisa dilihat di halaman [booking\\_dependencies](#)

## Class DatabaseWrapper

Setelah itu, kami dapat membuat sebuah Class bernama **`DatabaseWrapper`** yang berisi fungsi-fungsi yang akan digunakan dalam service class instances.

```


1  class DatabaseWrapper:
2
3      def __init__(self, connection):
4          self.connection = connection
5
6      def create_accident(self, id_booking, id_employee, description, compen
7          cursor = self.connection.cursor(dictionary=True)
8          sql = "INSERT INTO accident VALUES (NULL,{}, {},CURDATE(),'{}',{},{
9              id_booking, id_employee, description, compensation, compensati
10         cursor.execute(sql)
11         self.connection.commit()
12         cursor.close()
13         return "Add Accident Success"
14
15     def update_accident(self, id_accident, status):
16         cursor = self.connection.cursor(dictionary=True)
17         sql = "UPDATE accident SET status={} WHERE id = {}".format(
18             status, id_accident)
19         cursor.execute(sql)
20         self.connection.commit()
21         cursor.close()
22         return "Edit Accident Auccess"
23
24     def get_compensation(self, id_booking):
25         cursor = self.connection.cursor(dictionary=True)
26         sql = "SELECT compensation, compensation_cost FROM `accident` WHERE
27             id_booking)
28         cursor.execute(sql)
29         result = cursor.fetchone()
30         cursor.close()
31         return result

```

Fungsi 'create\_accident' berguna untuk membuat laporan accident baru jika terjadi masalah pada suatu booking dari customer. Misalnya adanya barang dalam kamar yang rusak

Fungsi 'update\_accident' berfungsi untuk mengupdate status dari suatu accident yang telah di catat sebelumnya, misalnya ingin mengupdate bahwa status accident sudah diselesaikan (masalah kelar)


Fungsi 'get\_compensation' berguna untuk melihat biaya kompensasi/denda dari suatu accident

 Penjelasan lebih lanjut tentang fungsi yang ada pada **accident service** dapat dilihat pada halaman RPC Accident

# CIRCULATION

## Import Libraries

```
1 import json
2 from nameko.rpc import rpc, RpcProxy
3 from nameko.events import EventDispatcher, event_handler
4 from nameko.web.handlers import http
```

 Penjelasan lebih lanjut tentang libraries diatas bisa dilihat pada halaman [booking\\_dependencies](#)

## Connect Microservice to Database

Contoh mengkoneksikan MySQL Database dengan microservice kami melalui dependencies :

 Untuk penjelasan bisa dilihat di halaman [booking\\_dependencies](#)

## Class DatabaseWrapper

Setelah itu, kami dapat membuat sebuah Class bernama **DatabaseWrapper** yang berisi fungsi-fungsi yang akan digunakan dalam service class instances.

```
1 class DatabaseWrapper:
2
3     def __init__(self, connection):
4         self.connection = connection
5
6     def add_circulation(self, id_room, id_item, id_employee, id_purchase,
7         cursor = self.connection.cursor(dictionary=True))
```

```

8         sql = "insert into circulation values (0,{}, {}, {}, {}, {}, {}, CURDATE())"
9         cursor.execute(sql)
10        self.connection.commit()
11        cursor.close()
12        if (self.get_room_item_by_id(id_room, id_item)) :
13            self.update_room_item(id_room, id_item, qty)
14        else :
15            self.add_room_item(id_room, id_item, qty)
16        return "sukses add circulation"
17
18    def get_room_item_by_id(self, id_room, id_item):
19        cursor = self.connection.cursor(dictionary=True)
20        sql = "SELECT * FROM room_item WHERE id_room = {} and id_item = {}"
21        cursor.execute(sql)
22        result = cursor.fetchone()
23        #print("Result:" + result)
24        cursor.close()
25        return result
26
27    def add_room_item(self, id_room, id_item, qty):
28        cursor = self.connection.cursor(dictionary=True)
29        sql = "insert into room_item values (0, {}, {}, {})".format(id_room, id_item, qty)
30        cursor.execute(sql)
31        self.connection.commit()
32        cursor.close()
33        return "sukses add room item"
34
35    def update_room_item(self, id_room, id_item, qty):
36        cursor = self.connection.cursor(dictionary=True)
37        cekqty = self.get_room_item_by_id(id_room, id_item)
38        #updated_qty = cekqty['qty'] + qty
39        #print(cekqty)
40        if ((cekqty['qty'] + qty) >= 0 and cekqty['qty'] is not None):
41            sql = "update room_item set qty = {} where id = {}".format(cekqty['qty'] + qty, id_item)
42            cursor.execute(sql)
43            self.connection.commit()
44            cursor.close()
45            return "sukses update qty room item"
46        else:
47            cursor.close()
48            return "Gagal update qty room item"
49

```

Fungsi '**add\_circulation**' berguna untuk menambah sirkulasi barang keluar atau masuk dari suatu kamar. Misalnya penambahan minuman soda atau cemilan

Fungsi '**get\_room\_item\_by\_id**' berguna untuk melihat semua data barang pada tabel **room\_item** berdasarkan kamar yang ingin dilihat




Penjelasan lebih lanjut tentang fungsi yang ada pada **circulation service** dapat dilihat pada halaman RPC Circulation



# PURCHASE ORDER

## Import Libraries

```
1 import json
2 from nameko.rpc import rpc, RpcProxy
3 from nameko.events import EventDispatcher, event_handler
4 from nameko.web.handlers import http
```

 Penjelasan lebih lanjut tentang libraries diatas bisa dilihat pada halaman [booking\\_dependencies](#)

## Connect Microservice to Database

Contoh mengkoneksikan MySQL Database dengan microservice kami melalui dependencies :

 Untuk penjelasan bisa dilihat di halaman [booking\\_dependencies](#)

## Class DatabaseWrapper

Setelah itu, kami dapat membuat sebuah Class bernama **DatabaseWrapper** yang berisi fungsi-fungsi yang akan digunakan dalam service class instances.

```
1 class DatabaseWrapper:
2
3     def __init__(self, connection):
4         self.connection = connection
5
6     def get_all_po(self):
7         cursor = self.connection.cursor(dictionary=True)
```


```

8         sql = "SELECT * FROM purchase_order"
9         cursor.execute(sql)
10        result = cursor.fetchall()
11        cursor.close()
12        return result
13
14    def create_po(self, id_employee, id_supplier, detail_purchase_order):
15        cursor = self.connection.cursor(dictionary=True)
16        sql = "INSERT INTO purchase_order VALUES (NULL,{},{},CURDATE(),1)"
17            id_employee, id_supplier)
18        cursor.execute(sql)
19        self.connection.commit()
20        cursor.close()
21        self.create_detail_po(detail_purchase_order)
22        return "Add purchase order success"

```

Fungsi '**get\_all\_po**' berguna untuk melihat semua data purchase order yang telah dilakukan


Fungsi '**create\_po**' berguna untuk membuat data purchase order baru

 Penjelasan lebih lanjut tentang fungsi yang ada pada **purchase order service** dapat dilihat pada halaman RPC Purchase Order

# ROOM


## Import Libraries

```
1 import json
2 from nameko.rpc import rpc, RpcProxy
3 from nameko.events import EventDispatcher, event_handler
4 from nameko.web.handlers import http
5
6 import pymysqlpool
7 import pymysql
8 from datetime import date
```

 Penjelasan lebih lanjut tentang libraries diatas bisa dilihat pada halaman [booking\\_dependencies](#)

## Connect Microservice to Database

Contoh mengkoneksikan MySQL Database dengan microservice kami melalui dependencies :

 Untuk penjelasan bisa dilihat di halaman [booking\\_dependencies](#)

## Class RoomWrapper

Setelah itu, kami dapat membuat sebuah Class bernama **RoomWrapper** yang berisi fungsi-fungsi yang akan digunakan dalam service class instances.

```
1 class RoomWrapper:
2     connection = None
3
```


```

4     def __init__(self, connection):
5         self.connection = connection
6
7     def get_room(self, id_room_type):
8         cursor = self.connection.cursor(pymysql.cursors.DictCursor)
9         sql = "SELECT * FROM `room` WHERE id_room_type = {}".format((id_ro
10        cursor.execute(sql)
11        return cursor.fetchall()
12
13    def delete_room_type(self, id):
14        cursor = self.connection.cursor(pymysql.cursors.DictCursor)
15        sql = 'DELETE FROM room_type WHERE id = {}'
16        sql = sql.format(id)
17        cursor.execute(sql)
18        return "Delete room type success."

```

Fungsi '**get\_room**' berguna untuk melihat semua data kamar pada hotel


Fungsi '**delete\_room\_type**' untuk menghapus tipe kamar yang tersedia pada hotel

 Penjelasan lebih lanjut tentang fungsi yang ada pada **room service** dapat dilihat pada halaman RPC Room

# STOCK MANAGEMENT

## Import Libraries

```
1 from nameko.extensions import DependencyProvider
2
3 import pymysqlpool
4 import pymysql
```

 Penjelasan lebih lanjut tentang libraries diatas bisa dilihat pada halaman [booking\\_dependencies](#)

## Connect Microservice to Database

Contoh mengkoneksikan MySQL Database dengan microservice kami melalui dependencies :

 Untuk penjelasan bisa dilihat di halaman [booking\\_dependencies](#)

## Class Item

Setelah itu, kami dapat membuat sebuah Class bernama **ITEM** yang berisi fungsi-fungsi yang akan digunakan dalam service class instances.

```
1 class Item:
2     connection = None
3
4     def __init__(self, connection):
5         self.connection = connection
6
7     def get_all_item(self):
```


```

8         cursor = self.connection.cursor(pymysql.cursors.DictCursor)
9         sql = 'SELECT * FROM item'
10        cursor.execute(sql)
11        return cursor.fetchall()
12
13    def insert_item(self, id_type, name, barcode, qty_in_hand, qty_broken,
14        result = {
15            'status': True,
16            'err_msg': ''
17        }
18
19        cursor = self.connection.cursor(pymysql.cursors.DictCursor)
20        sql = 'INSERT INTO item (id_type, name, barcode, qty_in_hand, qty_
21        sql = sql.format(id_type, name, barcode, qty_in_hand, qty_broken,
22        cursor.execute(sql)
23
24        return result

```

Fungsi '**get\_all\_item**' berguna untuk melihat semua data item yang disediakan oleh pihak hotel


Fungsi '**insert\_item**' untuk menambah data item dan qty nya yang tersedia

 Penjelasan lebih lanjut tentang fungsi yang ada pada **stock management service** dapat dilihat pada halaman RPC Stock Management

# SUPPLIER

## Import Libraries

```
1 import json
2 from nameko.rpc import rpc, RpcProxy
3 from nameko.events import EventDispatcher, event_handler
4 from nameko.web.handlers import http
```

 Penjelasan lebih lanjut tentang libraries diatas bisa dilihat pada halaman [booking\\_dependencies](#)

## Connect Microservice to Database

Contoh mengkoneksikan MySQL Database dengan microservice kami melalui dependencies :

 Untuk penjelasan bisa dilihat di halaman [booking\\_dependencies](#)

## Class DatabaseWrapper

Setelah itu, kami dapat membuat sebuah Class bernama **DatabaseWrapper** yang berisi fungsi-fungsi yang akan digunakan dalam service class instances.

```
1 class DatabaseWrapper:
2
3     def __init__(self, connection):
4         self.connection = connection
5
6     def get_all_supplier(self):
7         cursor = self.connection.cursor(dictionary=True)
```

```

8         result = []
9         sql = "SELECT * FROM supplier"
10        cursor.execute(sql)
11        for row in cursor.fetchall():
12            result.append({
13                'id': row['id'],
14                'name': row['name'],
15                'address': row['address'],
16                'phone_number2': row['phone_number2'],
17                'phone_number1': row['phone_number1'],
18                'email': row['email'],
19                'status': row['status'],
20                'last_update': row['last_update'],
21                'last_update_by': row['last_update_by']
22            })
23        cursor.close()
24        return result
25
26    def get_all_catalog(self):
27        cursor = self.connection.cursor(dictionary=True)
28        result = []
29        sql = "SELECT * FROM catalog"
30        cursor.execute(sql)
31        for row in cursor.fetchall():
32            result.append({
33                'id': row['id'],
34                'id_type': row['id_item'],
35                'id_supplier': row['id_supplier'],
36                'date': row['date'],
37                'unit': row['unit'],
38                'price_per_unit': row['price_per_unit'],
39                'status': row['status'],
40                'last_update': row['last_update'],
41                'last_update_by': row['last_update_by']
42            })
43        cursor.close()
44        return result

```

Fungsi '**get\_all\_supplier**' berguna untuk melihat semua data supplier yang bekerja sama dengan pihak hotel

Fungsi '**get\_all\_catalog**' berguna untuk melihat semua catalog item yang tersedia untuk dibeli






Penjelasan lebih lanjut tentang fungsi yang ada pada **supplier service** dapat dilihat pada halaman RPC Supplier

# EMPLOYEE MANAGEMENT

## Import Libraries

```
1 from nameko.extensions import DependencyProvider
2 import pymysqlpool
3 import pymysql
4 import smtplib, ssl
```

 Penjelasan lebih lanjut tentang libraries diatas bisa dilihat pada halaman [booking\\_dependencies](#)

## Connect Microservice to Database

Contoh mengkoneksikan MySQL Database dengan microservice kami melalui dependencies :

 Untuk penjelasan bisa dilihat di halaman [booking\\_dependencies](#)

## Class UserWrapper

Setelah itu, kami dapat membuat sebuah Class bernama **UserWrapper** yang berisi fungsi-fungsi yang akan digunakan dalam service class instances.

```
1 class UserWrapper:
2
3     connection = None
4
5     def __init__(self, connection):
6         self.connection = connection
7
```

```

8      def get_all_employee(self):
9          cursor = self.connection.cursor(pymysql.cursors.DictCursor)
10         sql = 'SELECT * FROM employee'
11         cursor.execute(sql)
12         return cursor.fetchall()
13
14     def edit_employee_data(self, id, name, birth, c_num, address, phone_num):
15         cursor = self.connection.cursor(pymysql.cursors.DictCursor)
16         sql = 'UPDATE employee SET name = "{}", date_of_birth = "{}", city = "{}", c_num = "{}", address = "{}", phone_num1 = "{}", phone_num2 = "{}" WHERE id = {}'.format(name, birth, c_num, address, phone_num1, phone_num2, id)
17         cursor.execute(sql)
18         return cursor.fetchone()
19

```

Fungsi '**get\_all\_employee**' berguna untuk melihat semua data karyawan yang bekerja

Fungsi '**edit\_employee\_data**' berguna untuk mengubah data suatu karyawan. Misalnya ingin merubah status jika sudah tidak bekerja lagi di hotel



Penjelasan lebih lanjut tentang fungsi yang ada pada **employee management service** dapat dilihat pada halaman RPC Employee Management

RPC

# BOOKING.py

## Import Libraries

```
from nameko.rpc import rpc
```

Disini harus meng-includekan rpc yang merupakan extension dari nameko. Dimana jika menggunakan extension ini harus meng-install rabbitMQ dan nameko terlebih dahulu



Download RabbitMQ = <https://www.rabbitmq.com/download.html>

pip install nameko -> dijalankan di CMD

## Import Dependencies File

```
1 import booking_dependencies
2 #from dependencies import booking_dependencies -> jika file dependencies b
```

Setelah itu akan meng-import dependencies dari booking service yang telah dibuat sebelumnya

```
1 class BookingService:
2
3     name = 'booking_service'
4
5     database = booking_dependencies.Database()
6
7     session = session_dependencies.SessionProvider()
8
9     @rpc
10    def get_customer(self, id=-1, ktp=-1):
```

```

11         customer = self.database.get_customer(id, ktp)
12         return customer
13
14
15     @rpc
16     def add_booking(self, id_customer, id_room_type, id_room, id_employee):
17         booking = self.database.add_booking(id_customer, id_room_type, id_
18         return booking
19
20
21     @rpc
22     def update_booking_room(self, id_booking, id_room_new, id_room_type_ne
23         updated_booking = self.database.update_booking_room(id_booking, id
24         return updated_booking
25
26
27     @rpc
28     def update_booking_date(self, session_id, id_booking, start_date, end_
29         err_msg = ''
30         stat = self.session.is_employee_online(session_id)
31         if stat:
32             data = self.session.get_session_data(session_id)
33             stat = self.database.get_booking_by_room(self.database.get_bo
34             if stat == True :
35                 updated_booking = self.database.update_booking_date(id_bo
36                 return updated_booking
37             else :
38                 return {
39                     "result": '',
40                     "err_msg": 'Pergantian tidak dapat dilakukan pada tang
41                     "status": False
42                 }
43         else:
44             err_msg = 'You are not logged in'
45             result = ''
46             status = False
47         return {
48             "result": result,
49             "err_msg": err_msg,
50             "status": status
51         }
52
53
54     @rpc
55     def update_booking_status(self, id_booking, status, id_employee):
56         updated_booking = self.database.update_booking_status(id_booking,
57         return updated_booking
58
59
60
61     @rpc

```

```

62     def get_booking(self, id_booking=-1, id_customer=-1):
63         booking = self.database.get_booking(id_booking, id_customer)
64         return booking
65
66
67     @rpc
68     def get_booking_by_room(self, id_room, start_date, end_date):
69         booking = self.database.get_booking_by_room(id_room, start_date, end_date)
70         return booking
71
72
73     @rpc
74     def get_detail_booking(self, id_booking=-1):
75         detail_booking = self.database.get_detail_booking(id_booking)
76         return detail_booking
77
78     @rpc
79     def add_detail_booking (self, id_service, id_booking, qty, price):
80         detail_booking = self.database.add_detail_booking(id_service, id_booking, qty, price)
81         return detail_booking
82
83     @rpc
84     def add_service (self, session_id, name, cost, status):
85         err_msg = ''
86         stat = self.session.is_employee_online(session_id)
87         if stat:
88             data = self.session.get_session_data(session_id)
89             service = self.database.add_service(name, cost, status, data['id_employee'])
90             err_msg = ''
91             result = service
92             status = True
93
94         else:
95             err_msg = 'You are not logged in'
96             result = ''
97             status = False
98         return {
99             "result": result,
100             "err_msg": err_msg,
101             "status": status
102         }
103
104     @rpc
105     def get_service(self, id_service=-1, service_name=-1):
106         service = self.database.get_service(id_service, service_name)
107         return service

```

Lalu kami membuat class BookingService dengan memberi nama = '**booking\_service**' yang nantinya berguna untuk pemanggilan oleh HTTP

```
database = booking_dependencies.Database()
```

Function diatas berguna untuk mengakses **class database** yang ada pada file **booking\_dependencies.py** agar rpc mengenal database yang akan dihubungkan.

```
session = session_dependencies.SessionProvider()
```

Session berguna untuk login dan logout. (Untuk pembahasan lebih dalam mengenai 'session' ada di dalam **employee service**)

## Functions in Booking Service

- **get\_customer** : fungsi ini akan menerima parameter berupa ID dari Customer dan/atau Nomor KTP customer setelah itu akan mereturnkan detail data dari customer tersebut
- **add\_booking** : fungsi ini untuk menambah reservasi yang dilakukan customer, dimana akan menerima parameter berupa id\_customer, id\_room\_type, id\_room, id\_employee, start\_date, end\_date, description, status dan
- **update booking date** : fungsi ini akan digunakan ketika salah seorang customer ingin meng-extend tanggal booking mereka, sistem akan mengecek apakah customer tersebut bisa memperpanjang waktu menginap mereka dihotel
- **update booking status** : fungsi ini berguna untuk mengupdate status booking
- **get\_booking** : fungsi ini akan meng returnkan data suatu booking customer
- **get\_booking by room** : fungsi ini untuk menampilkan semua booking berdasarkan kamar dimana fungsi ini akan menerima parameter id room dan range tanggal
- **get detail booking** : fungsi ini akan meng-returnkan detail suatu booking seperti service apa yang digunakan
- **add\_detail booking** : fungsi ini berguna jika salah seorang customer menggunakan service hotel seperti laundry atau makan didalam kamar. Fungsi ini menerima parameter berupa id service yang digunakan, nomor kamar, qty dan harganya
- **add\_service** : fungsi ini untuk menambah service yang disediakan hotel, dan menerima parameter berupa nama service yang ingin ditambahkan, harga, dan status service tersebut
- **get\_service** : untuk meng-returnkan detail salah satu service dan menerima parameter berupa id service dan/atau nama servicenya





## ORCHESTRATION

# BOOKING

## Import Libraries

```
1 from nameko.rpc import rpc, RpcProxy
2 from nameko.events import EventDispatcher, event_handler
3 import json, sys
```



Penjelasan lebih lanjut tentang libraries diatas bisa dilihat pada halaman [booking\\_dependencies](#)

## Class EntryDataBooking

Setelah itu, kami dapat membuat sebuah Class bernama **EntryDataBooking** yang berisi fungsi-fungsi yang akan digunakan dalam orchestration

```
1 class EntryDataBooking:
2     name = 'entry_booking_service'
3     room_service = RpcProxy('room_service')
4     booking_service = RpcProxy('booking_service')
5
6
7     session = session_dependencies.SessionProvider()
8
9     @rpc
10    def entry_booking(self, session_id, room_type_name, start_date, end_da
11        err_msg = ''
12        stat = self.session.is_employee_online(session_id)
13        if stat:
14            data = self.session.get_session_data(session_id)
15
16            id_room_type = self.room_service.get_room_type_by_name(room_ty
17            room = self.room_service.get_room_by_type(id_room_type)
18
19            check = False
20            for room_info in room:
21                print (room_info['id'])
```

```

22         check_booking_sebelumnya = self.booking_service.get_booking
23         if check_booking_sebelumnya == True:
24             insert_data = self.booking_service.add_booking(1, id_r
25             for service in services:
26                 service_from_db = self.booking_service.get_service
27                 price = service_from_db['cost']
28                 self.booking_service.add_detail_booking(service_fr
29
30             check = True
31             break
32
33         if check == False:
34             err_msg = 'Maaf, kamar tidak tersedia untuk saat ini'
35             result = ''
36             status = False
37
38         elif check == True:
39             err_msg = ''
40             result = insert_data
41             status = True
42
43     else:
44         err_msg = 'You are not logged in'
45         result = ''
46         status = False
47     return {
48         "result": result,
49         "err_msg": err_msg,
50         "status": status
51     }

```

Dalam fungsi '**entry\_booking**' akan menerima parameter berupa session\_id, room\_type\_name, start\_date, end\_date, description, services yang diperlukan untuk membuat data booking baru. Setelah itu akan dilakukan pengecekan status dari **session id**. Jika session id kosong maka akan keluar '**error msg**' bahwa belum login, sebaliknya jika session id ada maka akan dilakukan penginputan data booking

```

1  @rpc
2      def swap_booking_room(self, session_id, id_booking):
3          err_msg = ''
4          stat = self.session.is_employee_online(session_id)
5          booking_data = self.booking_service.get_booking(id_booking = id_bo
6          if booking_data['status'] <= 0:
7              if stat:
8                  data = self.session.get_session_data(session_id)

```

```

9         id_room_type = self.booking_service.get_booking(id_booking
10         start_date = booking_data['start_date']
11         end_date = booking_data['end_date']
12         room = self.room_service.get_room_by_type(id_room_type)
13
14         check = False
15         for room_info in room:
16             print (room_info['id'])
17             check_booking_sebelumnya = self.booking_service.get_bo
18             if check_booking_sebelumnya == True:
19                 update_data = self.booking_service.update_booking_
20                 check = True
21                 break
22
23         if check == False:
24             result = ""
25             err_msg = "Maaf, penukaran kamar tidak tersedia untuk
26             status = False
27             # return {'status': 0, 'message': 'Maaf, penukaran kam
28         elif check == True:
29             result = update_data
30             err_msg = ""
31             status = True
32             # return update_data
33
34         else:
35             err_msg = 'You are not logged in'
36             result = ''
37             status = False
38     else:
39         err_msg = "Pelanggan yang bersangkutan sudah menginap atau tel
40         result = ''
41         status = False
42
43     return {
44         "result": result,
45         "err_msg": err_msg,
46         "status": status
47     }

```

Untuk fungsi 'swap\_booking\_room' akan menerima parameter 'session\_id' dan 'id\_booking' yang mana yang ingin ditukarkan. Sistem akan mengecek tanggal booking yang ingin ditukarkan apakah kamar tersedia atau tidak. Jika sistem menemukan kamar yang bisa ditukarkan dengan **booking yang lama** maka customer akan mendapat pergantian kamar dan mengupdate status booking dan kamar yang ditukarkan.

```

1  @rpc
2      def update_booking_room(self, session_id, id_booking, room_type_new):
3          err_msg = ''
4          stat = self.session.is_employee_online(session_id)
5
6          booking_data = self.booking_service.get_booking(id_booking = id_bo
7
8          if booking_data['status'] <= 0:
9
10             if stat:
11                 data = self.session.get_session_data(session_id)
12                 id_room_type = self.room_service.get_room_type_by_name(roo
13                 room = self.room_service.get_room_by_type(id_room_type)
14
15                 start_date = booking_data['start_date']
16                 end_date = booking_data['end_date']
17
18                 check = False
19                 for room_info in room:
20                     print (room_info['id'])
21                     check_booking_sebelumnya = self.booking_service.get_bo
22                     if check_booking_sebelumnya == True:
23                         update_data = self.booking_service.update_booking_
24                         check = True
25                         break
26
27                 if check == False:
28                     err_msg = 'Maaf, penukaran tipe kamar tidak tersedia u
29                     result = ''
30                     status = False
31                 elif check == True:
32                     err_msg = ''
33                     result = update_data
34                     status = True
35             else:
36                 err_msg = 'You are not logged in'
37                 result = ''
38                 status = False
39         else:
40             err_msg = "Pelanggan yang bersangkutan sudah menginap atau tel
41             result = ""
42             status = False
43         return {
44             "result": result,
45             "err_msg": err_msg,
46             "status": status
47     }

```

Untuk fungsi '**update\_booking\_room**' akan menerima parameter '**id\_booking**' dan '**room\_type\_new**'. Fungsi ini digunakan ketika customer ingin meng-upgrade tipe kamarnya misalnya dari **standard room** ke **luxury room**, maka sistem akan mengecek mulai dari apakah kamar yang diinginkan ditempati oleh customer lain atau tidak, jika tidak maka customer bisa meng-upgrade tipe kamar pesanan mereka.

```
1  @rpc
2      def check_order_review(self, ktp):
3
4          customer_data = self.booking_service.get_customer(ktp = ktp)
5          id_customer = customer_data['id']
6          customer_name = customer_data['name']
7
8          result = {
9              "cust_id": id_customer,
10             "cust_name": customer_name,
11             "booking": []
12         }
13
14         booking_data = self.booking_service.get_booking(id_customer = id_c
15 i = 0
16         for booking in booking_data:
17             room_number = self.room_service.get_room_num(booking['id_room']
18             room_type = self.room_service.get_room_type_by_id(booking['id_
19             result['booking'].append(
20                 {
21                     "room_number": room_number,
22
23                     "room_type": room_type,
24                     "booking_date": booking['booking_date'],
25                     "start_date": booking['start_date'],
26                     "end_date": booking['end_date'],
27                     "description": booking['description'],
28                     "services": []
29                 }
30             )
31             detail_booking = self.booking_service.get_detail_booking(id_bo
32
33             for detail in detail_booking:
34                 service_data = self.booking_service.get_service(id_service
35                 result['booking'][i]['services'].append(
36                     {
37                         "service_name": service_data['name']
38                     }
39                 )
40
41             i = i+1
```

```
42
```

```
43     return result
```

Untuk fungsi '**check\_order\_review**' akan menerima parameter ktp dari customer yang ingin dilihat lagi detail ordernya. Fungsi ini akan meng-returnkan mulai dari data pelanggan hingga detail dari booking pelanggan tersebut seperti service apa yang digunakan selama menginap di hotel.