|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | |  |
|  | | | |  | | |  | | |
|  | ФГАОУ ВО «Пермский государственный национальный исследовательский университет» | | | | | | | |  |
|  | | | |  | | |  | | |
|  | | | ОТЧЕТ  О ВЫПОЛНЕНИИ ЗАДАНИЯ №4  ПО ДИСЦИПЛИНЕ ТРРП  по теме:  «Разработка распределенного приложения  для локальной сети» | | | | |  | |
|  | | | |  | | |  | | |
|  | | Работу выполнили  студенты гр. ФИТ-1,2-21,  3 курс  Кузнецов А. М. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  и Голдобин Д. А.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_\_» \_\_\_\_\_\_\_ 2023 г. | | |  | Проверил  ст. преподаватель  кафедры МОВС  Постаногов И. С.  \_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_ 2023 г. | | |  |
|  | | | | Пермь 2023 | | |  | | |

# 1. Постановка задачи

Требования к выполнению работы:

1. Приложение должно обеспечивать параллельную работу нескольких клиентов и серверов. Дополнительное требование: возможность запуска нескольких серверов на одном компьютере.
2. Клиентские приложения должны автоматически находить серверы для обслуживания и выполнения заданных функций.
3. Серверы системы могут выполнять различные функции.
4. При разрыве сеанса приложения должны автоматически восстанавливать свою работоспособность.
5. Приложения должны поддерживать возможность взаимодействия в различных режимах.
6. Для организации взаимодействия нужно использовать различные средства коммуникации, сравнив их возможности.

# 2. Общее описание приложения и предметной области.

Предметная область – бронирование комнат в хостелах или в других местах. Клиент может просмотреть список всех комнат и их статус, а также забронировать комнату на своё имя.

# 3. Архитектура системы.Изображение выглядит как диаграмма, линия, План, График Автоматически созданное описание

Система состоит из клиентов, одного диспетчера, серверов бронирования, серверов управления местами и БД под управлением PostgreSQL, расположенной на виртуальной машине на платформе Yandex Cloud.

Данная архитектура была выбрана из-за:

* возможности масштабирования серверов;
* отказоустойчивости при выходе из строя какого-либо из серверов любого вида;
* относительной простоты реализации одного диспетчера.

# 4. Алгоритм работы приложения

1. Все узлы запускаются и настраиваются с помощью соответствующих конфигурационных файлов, где указаны адреса подключения каждого из узлов, взаимодействие между ними происходит при помощи gRPC.
2. Клиент может отправить один из двух запросов диспетчеру: просмотр списка комнат и их статус, бронирование определенной комнаты на имя клиента.
3. Если клиент отправляет запрос на просмотр комнат диспетчеру, то диспетчер пересылает этот запрос одному из серверов управления местами, который уже просмотрит БД и вернёт результат в виде списка мест обратно диспетчеру, а тот клиенту.
4. Если клиент отправляет запрос на бронирование комнаты, где указывает номер комнаты и своё имя, диспетчеру, то диспетчер пересылает этот запрос одному из серверов бронирования, который пересылает запрос одному из серверов управления местами. Сервер управления местами просматривает БД и определяет, свободна ли выбранная комната. Если свободна, то комната бронируется, и клиенту через сервер бронирования и диспетчер отправляется ответ об успешном бронировании. Если комната была занята, то таким же образом отправляется ответ о невозможности бронирования данной комнаты.
5. Если в конфигурационных файлах описано более одного сервера любого вида, то распределение запросов между ними определяется круговым порядком.
6. Если какой-то узел становится недоступен, то клиенту будет сообщено о недоступности узла.
7. Доступность диспетчера для клиента производится в отдельном потоке раз в секунду. Доступность серверов для диспетчера, серверов управления местами для серверов бронирования также происходит в отдельных потоках.

# 4. Методы коммуникаций компонентов системы

Вся коммуникация в системе осуществляется через удаленные вызовы процедур, предоставляемые gRPC.

# 5. Способ передачи данных

В данном приложении используется однонаправленное взаимодействие между всеми узлами, кроме сервера управления местами сервером бронирования используется потоковая передача сервера (в ответ на запрос сервера бронирования сервер управления местами предоставляет поток сообщений). Так же все вызовы синхронны, т.е. каждый из узлов ждёт ответа от другого узла перед тем как выполнить следующее действие. Данный способ передачи выбран из-за простоты реализации.

# 6. Структура передаваемых данных

Структура передаваемых данных описана в файле reservation.proto

syntax = "proto3";

package reservation;

// Сервис бронирования

service ReservationService {

rpc Ping (PingRequest) returns (PingResponse);

rpc ReserveSeat (ReservationRequest) returns (ReservationResponse);

rpc GetSeatStatus (SeatListRequest) returns (stream SeatStatus);

}

// Запрос проверки соединения

message PingRequest {}

// Ответ на проверку соединения

message PingResponse {}

//Запрос бронирования

message ReservationRequest {

string customer\_name = 1;

int32 seat\_number = 2;

}

//Ответ на запрос бронирования

message ReservationResponse {

bool success = 1;

string message = 2;

}

//Запрос на просмотр всех мест

message SeatListRequest {}

//Ответ на запрос просмотра

message SeatStatus {

int32 seat\_number = 1;

string status = 2;

}

# 7. Отказоустойчивость системы

При выходе из строя одного из элементов системы, клиенту будут выводиться соответствующие сообщения о недоступности одного из узлов. При восстановлении вышедшего из строя узла остальные элементы системы не нужно будет перезагружать, и система продолжит работу.

Если из строя выйдет диспетчер, то клиенту будет выведено соответствующее сообщение. Когда диспетчер вновь восстановится, то клиент через несколько секунд обнаружит его и сможет снова посылать ему запросы.

# Работа с БД

Комнаты, их статус, и имена забронировавшего хранятся в БД, расположенной на виртуальной машине в Yandex Cloud. Сама БД представлена одной таблицей rooms следующей структуры.

|  |  |  |  |
| --- | --- | --- | --- |
| roomid | roomname | roomstatus | guestname |

В БД добавлены несколько комнат со статусом «Свободно», а поле guestname содержит NULL.

# 9. Исходный код приложений с комментариями.

Клиент:

import grpc  
import time  
import threading  
import reservation\_pb2  
import reservation\_pb2\_grpc  
  
class Client:  
 def \_\_init\_\_(self, config\_file):  
 *# чтение конфигурационного файла при инициализации клиента* self.config = self.read\_config(config\_file)  
 *# канал для соединения с диспетчером* self.dispatcher\_channel = grpc.insecure\_channel(  
 f"{self.config['DispatcherServerIP']}:{self.config['DispatcherServerPort']}"  
 )  
 *# блокировки и события* self.lock = threading.Lock()  
 self.dispatcher\_available = True  
 self.result\_received\_event = threading.Event()  
  
 def read\_config(self, config\_file):  
 *# чтение конфигурационного файла и возвращение словаря с настройками* config = {}  
 with open(config\_file, 'r') as file:  
 for line in file:  
 key, value = map(str.strip, line.split('='))  
 config[key] = value  
 return config  
  
 def check\_dispatcher\_availability(self):  
 try:  
 *# проверка доступности диспетчера* stub = reservation\_pb2\_grpc.ReservationServiceStub(self.dispatcher\_channel)  
 ping\_request = reservation\_pb2.PingRequest()  
 response = stub.Ping(ping\_request)  
 return True  
 except grpc.RpcError as e:  
 return False  
  
 def check\_dispatcher\_thread(self):  
 *# периодическая проверка доступности диспетчера в отдельном потоке* while True:  
 with self.lock:  
 self.dispatcher\_available = self.check\_dispatcher\_availability()  
 if self.dispatcher\_available:  
 self.result\_received\_event.set()  
 time.sleep(1)  
  
 def reserve\_seat(self, customer\_name, seat\_number):  
 *# отправка запроса на бронирование места через диспетчер* stub = reservation\_pb2\_grpc.ReservationServiceStub(self.dispatcher\_channel)  
 reservation\_request = reservation\_pb2.ReservationRequest(  
 customer\_name=customer\_name,  
 seat\_number=seat\_number  
 )  
 response = stub.ReserveSeat(reservation\_request)  
 print(f"Успешность: {response.success}, Сообщение: {response.message}")  
 self.result\_received\_event.set()  
  
 def get\_seat\_status(self):  
 *# получение статуса всех мест через диспетчер* stub = reservation\_pb2\_grpc.ReservationServiceStub(self.dispatcher\_channel)  
 seat\_status\_request = reservation\_pb2.SeatListRequest()  
 seat\_status\_response = stub.GetSeatStatus(seat\_status\_request)  
 for status in seat\_status\_response:  
 print(f"Место {status.seat\_number}: {status.status}")  
 self.result\_received\_event.set()  
  
 def start(self):  
 *# запуск потока для проверки доступности диспетчера* thread = threading.Thread(target=self.check\_dispatcher\_thread, daemon=True)  
 thread.start()  
  
 try:  
 while True:  
 with self.lock:  
 if self.dispatcher\_available:  
 print("1. Просмотр всех комнат и их статус")  
 print("2. Забронировать комнату")  
 print("3. Выход из программы")  
 choice = input("Выберите действие: ")  
  
 if choice == "1":  
 *# вывод статуса всех мест через диспетчер* try:  
 self.get\_seat\_status()  
 self.result\_received\_event.wait()  
 self.result\_received\_event.clear()  
 except grpc.RpcError as e:  
 print("Один из узлов сети недоступен. Пожалуйста, подождите...")  
 elif choice == "2":  
 *# отправка запроса на бронирование места через диспетчер* customer\_name = input("Введите ваше имя: ")  
 seat\_number = int(input("Введите номер места для бронирования: "))  
 try:  
 self.reserve\_seat(customer\_name, seat\_number)  
 self.result\_received\_event.wait()  
 self.result\_received\_event.clear()  
 except grpc.RpcError as e:  
 print("Один из узлов сети недоступен. Пожалуйста, подождите...")  
 elif choice == "3":  
 print("Программа завершена.")  
 break  
 else:  
 print("Некорректный выбор. Пожалуйста, выберите 1, 2 или 3.")  
 else:  
 print("Диспетчер недоступен. Подождите, пожалуйста...")  
 self.result\_received\_event.clear()  
 time.sleep(5)  
 except KeyboardInterrupt:  
 print("Клиент завершен")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 client = Client("config.txt")  
 client.start()

Диспетчер:

import grpc  
from concurrent import futures  
import reservation\_pb2  
import reservation\_pb2\_grpc  
import threading  
import time  
  
def read\_config(config\_name):  
 *# чтение конфигурационного файла и возвращение словаря с настройками* config = {}  
 with open(config\_name, 'r') as file:  
 for line in file:  
 key, value = map(str.strip, line.split('='))  
 config[key] = value  
 return config  
  
class DispatcherServicer(reservation\_pb2\_grpc.ReservationServiceServicer):  
 def \_\_init\_\_(self):  
 self.reservation\_stubs = []  
 self.seat\_management\_stubs = []  
 self.reservation\_channels = []  
 self.seat\_management\_channels = []  
 self.reservation\_stub\_lock = threading.Lock()  
 self.seat\_management\_stub\_lock = threading.Lock()  
 self.current\_reservation\_server = 0  
 self.current\_seat\_management\_server = 0  
 self.server\_statuses = []  
  
 *# создание каналов для серверов бронирования и управления местами* for i in range(int(config['ReservationServerCount'])):  
 self.add\_server\_channel(  
 i + 1,  
 f"ReservationServerIP{i + 1}",  
 f"ReservationServerPort{i + 1}",  
 self.reservation\_channels,  
 self.reservation\_stubs,  
 self.server\_statuses  
 )  
  
 for i in range(int(config['SeatManagementServerCount'])):  
 self.add\_server\_channel(  
 i + 1,  
 f"SeatManagementServerIP{i + 1}",  
 f"SeatManagementServerPort{i + 1}",  
 self.seat\_management\_channels,  
 self.seat\_management\_stubs,  
 self.server\_statuses  
 )  
  
 def add\_server\_channel(self, server\_number, ip\_key, port\_key, channels, stubs, statuses):  
 *# добавление канала и стаба для сервера* server\_ip = config[ip\_key]  
 server\_port = int(config[port\_key])  
 channel = grpc.insecure\_channel(f"{server\_ip}:{server\_port}")  
 stub = reservation\_pb2\_grpc.ReservationServiceStub(channel)  
 channels.append(channel)  
 stubs.append(stub)  
 statuses.append(True)  
  
 def get\_next\_active\_reservation\_server(self):  
 *# получение следующего активного сервера бронирования* with self.reservation\_stub\_lock:  
 active\_reservation\_servers = [i for i, status in  
 enumerate(self.server\_statuses[:int(config['ReservationServerCount'])])  
 if status]  
  
 if not active\_reservation\_servers:  
 return None  
  
 self.current\_reservation\_server = (self.current\_reservation\_server + 1) % len(active\_reservation\_servers)  
 return active\_reservation\_servers[self.current\_reservation\_server]  
  
 def get\_next\_active\_seat\_management\_server(self):  
 *# получение следующего активного сервера управления местами* with self.seat\_management\_stub\_lock:  
 active\_seat\_management\_servers = [i for i, status in  
 enumerate(self.server\_statuses[int(config['ReservationServerCount']):])  
 if status]  
  
 if not active\_seat\_management\_servers:  
 return None  
  
 self.current\_seat\_management\_server = (self.current\_seat\_management\_server + 1) % len(active\_seat\_management\_servers)  
 return active\_seat\_management\_servers[self.current\_seat\_management\_server]  
  
 def check\_server\_availability(self, server\_number, channels, statuses):  
 *# проверка доступности сервера в отдельном потоке* while True:  
 time.sleep(1)  
 with self.reservation\_stub\_lock:  
 channel = channels[server\_number - 1]  
 try:  
 grpc.channel\_ready\_future(channel).result(timeout=1)  
 statuses[server\_number - 1] = True  
 except grpc.FutureTimeoutError:  
 statuses[server\_number - 1] = False  
  
 def start\_server\_availability\_check(self, server\_number, channels, statuses):  
 *# запуск потока для проверки доступности сервера* thread = threading.Thread(  
 target=self.check\_server\_availability,  
 args=(server\_number, channels, statuses),  
 daemon=True  
 )  
 thread.start()  
  
 def Ping(self, request, context):  
 *# проверка доступности диспетчера* return reservation\_pb2.PingResponse()  
  
 def ReserveSeat(self, request, context):  
 active\_reservation\_server = self.get\_next\_active\_reservation\_server()  
 if active\_reservation\_server is None:  
 return reservation\_pb2.ReservationResponse(  
 success=False,  
 message="Извините, все сервера бронирования недоступны. Попробуйте позже."  
 )  
  
 stub = self.reservation\_stubs[active\_reservation\_server]  
 reservation\_response = stub.ReserveSeat(request)  
 return reservation\_response  
  
 def GetSeatStatus(self, request, context):  
 active\_seat\_management\_server = self.get\_next\_active\_seat\_management\_server()  
 if active\_seat\_management\_server is None:  
 return reservation\_pb2.SeatStatusResponse(  
 success=False,  
 message="Извините, все сервера управления местами недоступны. Попробуйте позже."  
 )  
  
 stub = self.seat\_management\_stubs[active\_seat\_management\_server]  
 seat\_status\_response = stub.GetSeatStatus(request)  
 return seat\_status\_response  
  
def serve\_dispatcher():  
 global config  
 config = read\_config("confdispatch.txt")  
  
 server = grpc.server(futures.ThreadPoolExecutor(max\_workers=10))  
 dispatcher\_servicer = DispatcherServicer()  
 reservation\_pb2\_grpc.add\_ReservationServiceServicer\_to\_server(dispatcher\_servicer, server)  
  
 for i in range(int(config['ReservationServerCount'])):  
 dispatcher\_servicer.start\_server\_availability\_check(i + 1, dispatcher\_servicer.reservation\_channels, dispatcher\_servicer.server\_statuses)  
  
 for i in range(int(config['SeatManagementServerCount'])):  
 dispatcher\_servicer.start\_server\_availability\_check(i + 1, dispatcher\_servicer.seat\_management\_channels, dispatcher\_servicer.server\_statuses)  
  
 server.add\_insecure\_port(f"{config['IP']}:{config['Port']}")  
 *# запуск диспетчера* server.start()  
 print(f"Диспетчер запущен и слушает на {config['IP']}:{config['Port']}...")  
 try:  
 server.wait\_for\_termination()  
 except KeyboardInterrupt:  
 print("Диспетчер завершает работу")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 serve\_dispatcher()

Сервер управления местами:

import grpc  
from concurrent import futures  
import reservation\_pb2  
import reservation\_pb2\_grpc  
import threading  
import time  
import psycopg2  
  
def read\_config(config\_name):  
 *# чтение конфигурационного файла и возвращение словаря с настройками* config = {}  
 with open(config\_name, 'r') as file:  
 for line in file:  
 key, value = map(str.strip, line.split('='))  
 config[key] = value  
 return config  
class SeatManager(reservation\_pb2\_grpc.ReservationServiceServicer):  
 def \_\_init\_\_(self, db\_connection):  
 *# инициализация SeatManager с подключением к базе данных* self.db\_connection = db\_connection  
  
 def Ping(self, request, context):  
 return reservation\_pb2.PingResponse()  
  
 def ReserveSeat(self, request, context):  
 room\_id\_to\_book = request.seat\_number  
 guest\_name\_to\_set = request.customer\_name  
  
 cursor = self.db\_connection.cursor()  
  
 try:  
 *# проверка текущего состояния комнаты в базе данных и выполнение бронирования* cursor.execute("SELECT roomstatus FROM Rooms WHERE roomid = %s;", (room\_id\_to\_book,))  
 current\_status = cursor.fetchone()  
  
 if current\_status and not current\_status[0]:  
 cursor.execute("UPDATE Rooms SET roomstatus = true, guestname = %s WHERE roomid = %s;",  
 (guest\_name\_to\_set, room\_id\_to\_book))  
 self.db\_connection.commit()  
 response\_success = True  
 response\_message = f"Комната {room\_id\_to\_book} успешно забронирована для {guest\_name\_to\_set}."  
 else:  
 response\_success = False  
 response\_message = f"Комната {room\_id\_to\_book} уже занята."  
  
 except Exception as e:  
 response\_success = False  
 response\_message = f"Ошибка: {e}"  
  
 finally:  
 cursor.close()  
  
 return reservation\_pb2.ReservationResponse(success=response\_success, message=response\_message)  
  
 def GetSeatStatus(self, request, context):  
 cursor = self.db\_connection.cursor()  
  
 try:  
 cursor.execute("SELECT roomid, roomname, CASE WHEN roomstatus THEN 'Занято' ELSE 'Свободно' END, "  
 "CASE WHEN roomstatus THEN guestname ELSE NULL END FROM Rooms;")  
 all\_rooms = cursor.fetchall()  
  
 for room in all\_rooms:  
 room\_id, room\_name, status, guest\_name = room  
 yield reservation\_pb2.SeatStatus(seat\_number=room\_id, status=status)  
  
 except Exception as e:  
 print(f"Ошибка: {e}")  
  
 finally:  
 cursor.close()  
  
def serve\_reservation():  
 config = read\_config("confseatmanserv.txt")  
  
 *# подключение к облачной базе данных* db\_connection = psycopg2.connect("""  
 host=rc1d-5g27m0aeolofwslv.mdb.yandexcloud.net  
 port=6432  
 dbname=db1  
 user=user1  
 password=kavalsky945  
 target\_session\_attrs=read-write  
 """)  
  
 server = grpc.server(futures.ThreadPoolExecutor(max\_workers=10))  
  
 reservation\_pb2\_grpc.add\_ReservationServiceServicer\_to\_server(  
 SeatManager(db\_connection), server  
 )  
  
 server.add\_insecure\_port(f"{config['IP']}:{config['Port']}")  
 *# запуск сервера* server.start()  
 print(f"Сервер управления местами запущен и слушает на {config['IP']}:{config['Port']}...")  
  
 try:  
 server.wait\_for\_termination()  
  
 except KeyboardInterrupt:  
 print("Сервер управления местами завершает работу")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 serve\_reservation()

Сервер бронирования:

import grpc  
from concurrent import futures  
import reservation\_pb2  
import reservation\_pb2\_grpc  
import threading  
import time  
  
def read\_config(config\_name):  
 *# чтение конфигурационного файла и возвращение словаря с настройками* config = {}  
 with open(config\_name, 'r') as file:  
 for line in file:  
 key, value = map(str.strip, line.split('='))  
 config[key] = value  
 return config  
  
class ReservationServerServicer(reservation\_pb2\_grpc.ReservationServiceServicer):  
 def \_\_init\_\_(self):  
 *# инициализация ReservationServerServicer с каналами и блокировками* self.seat\_management\_stubs = []  
 self.seat\_management\_channels = []  
 self.seat\_management\_stub\_lock = threading.Lock()  
 self.current\_seat\_management\_server = 0  
 self.server\_statuses = []  
  
 *# Создаем каналы здесь* for i in range(int(config['SeatManagementServerCount'])):  
 self.add\_server\_channel(  
 i + 1,  
 f"SeatManagementServerIP{i + 1}",  
 f"SeatManagementServerPort{i + 1}",  
 self.seat\_management\_channels,  
 self.seat\_management\_stubs,  
 self.server\_statuses  
 )  
  
 def add\_server\_channel(self, server\_number, ip\_key, port\_key, channels, stubs, statuses):  
 server\_ip = config[ip\_key]  
 server\_port = int(config[port\_key])  
 channel = grpc.insecure\_channel(f"{server\_ip}:{server\_port}")  
 stub = reservation\_pb2\_grpc.ReservationServiceStub(channel)  
 channels.append(channel)  
 stubs.append(stub)  
 statuses.append(True) *# Изначально считаем сервера неактивными* def get\_next\_active\_seat\_management\_server(self):  
 *# получение следующего активного сервера управления местами* with self.seat\_management\_stub\_lock:  
 active\_seat\_management\_servers = [i for i, status in  
 enumerate(self.server\_statuses)  
 if status]  
  
 if not active\_seat\_management\_servers:  
 return None  
  
 self.current\_seat\_management\_server = (self.current\_seat\_management\_server + 1) % len(active\_seat\_management\_servers)  
 return active\_seat\_management\_servers[self.current\_seat\_management\_server]  
  
 def check\_server\_availability(self, server\_number, channels, statuses):  
 while True:  
 *# периодическая проверка доступности сервера управления местами* time.sleep(1)  
 with self.seat\_management\_stub\_lock:  
 channel = channels[server\_number - 1]  
 try:  
 grpc.channel\_ready\_future(channel).result(timeout=1)  
 statuses[server\_number - 1] = True  
 except grpc.FutureTimeoutError:  
 statuses[server\_number - 1] = False  
  
 def start\_server\_availability\_check(self, server\_number, channels, statuses):  
 *# запуск потока для проверки доступности сервера управления местами* thread = threading.Thread(  
 target=self.check\_server\_availability,  
 args=(server\_number, channels, statuses),  
 daemon=True  
 )  
 thread.start()  
  
 def ReserveSeat(self, request, context):  
 *# бронирование мест* active\_seat\_management\_server = self.get\_next\_active\_seat\_management\_server()  
 if active\_seat\_management\_server is None:  
 return reservation\_pb2.ReservationResponse(  
 success=False,  
 message="Извините, все сервера управления местами недоступны. Попробуйте позже"  
 )  
  
 stub = self.seat\_management\_stubs[active\_seat\_management\_server]  
 seat\_status\_response = stub.ReserveSeat(request)  
 seat\_status\_response.message+=f" (север бронирования 1)"  
 return seat\_status\_response  
  
def serve\_reservation\_server():  
 global config  
 config = read\_config("confreserv.txt")  
  
 server = grpc.server(futures.ThreadPoolExecutor(max\_workers=10))  
 reservation\_servicer = ReservationServerServicer()  
 reservation\_pb2\_grpc.add\_ReservationServiceServicer\_to\_server(reservation\_servicer, server)  
 *# запуск проверки доступности серверов управления местами* for i in range(int(config['SeatManagementServerCount'])):  
 reservation\_servicer.start\_server\_availability\_check(i + 1, reservation\_servicer.seat\_management\_channels, reservation\_servicer.server\_statuses)  
 *# запуск сервера* server.add\_insecure\_port(f"{config['IP']}:{config['Port']}")  
 server.start()  
 print(f"Сервер бронирования запущен и слушает на {config['IP']}:{config['Port']}...")  
 try:  
 server.wait\_for\_termination()  
 except KeyboardInterrupt:  
 print("Сервер бронирования завершает работу")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 serve\_reservation\_server()