Endpoints:
in admin Service

```
public void approveCompany(Integer adminId,Integer
companyId){
    Admin admin = adminRepository.findAdminById(adminId);
    Company company =
companyRepository.findCompanyById(companyId);
    if(admin==null||company==null){
        throw new ApiException("Can't approve");
    }
    if(!company.getIsApproved()){
        company.setIsApproved(true);
        companyRepository.save(company);
    }else throw new ApiException("Company is already
approved");


}

//Raghad
public void approveExpert(Integer adminId,Integer
expertId){
    MaintenanceExpert expert =
maintenanceExpertRepository.findMaintenanceExpertById(exp
ertId);
    Admin admin = adminRepository.findAdminById(adminId);
    if(expert ==null|| admin==null){
        throw new ApiException("Can't approve");
    }

    if(!expert.getIsApproved()){
        expert.setIsApproved(true);
        maintenanceExpertRepository.save(expert);
    }else throw new ApiException("Maintenance Expert is
already approved");
}
```

filterCourses in Course Service

```
//Raghad
   public List<CourseDTO> filterCourses(Double minPrice,
Double maxPrice, Integer minDuration, Integer
maxDuration) {
       // Fetch filtered courses from the repository
```

```
        List<Course> courses =
courseRepository.filterCourses(minPrice, maxPrice,
minDuration, maxDuration);

        // Map courses to CourseDTOs
        return courses.stream().map(course -> new
CourseDTO(
                course.getName(),
                course.getDescription(),
                course.getPrice(),
                course.getDuration() // Include trainer
name in the DTO
        )).collect(Collectors.toList());
    }
```

in Fine Service

```
    //Raghad
    //  Fetch all fines by user ID
    public List<FineDTO> getAllFineByUserId(Integer
userId){
        List<Fine> fines =
fineRepository.findFineByUserId(userId);
        List<FineDTO> fineDTOS=new ArrayList<>();

        for(Fine fine:fines){
            FineDTO fineDTO = new
FineDTO(fine.getDescription(),fine.getAmount(),fine.getIs
Paid());
            fineDTOS.add(fineDTO);
        }
        return fineDTOS;
    }

//Raghad
// Scheduled job to impose late return penalties
@Scheduled(cron = "0 0 0 * * *") // Run daily at midnight
//@Scheduled(fixedRate = 60000)
public void imposeLateReturnPenalties() {
    // Fetch overdue renting requests where the bike is
not yet returned
    List<RentingRequest> overdueRequests =
rentingRequestRepository.findOverdueRentals(LocalDate.now
());
```

```
   for (RentingRequest rentingRequest : overdueRequests)
{
       // Skip fine calculation if the bike has been
returned
       if
(Boolean.TRUE.equals(rentingRequest.getIsReturned())) {
           continue;
       }

       LocalDate endDate = rentingRequest.getEndDate();
       LocalDate currentDate = LocalDate.now();

       // Calculate late days
       long lateDays =
java.time.temporal.ChronoUnit.DAYS.between(endDate,
currentDate);

       if (lateDays > 0) {
           // Calculate fine amount
           Renting renting = rentingRequest.getRenting();
           if (renting == null) {
               System.err.println("Renting details not
found for RentingRequest ID " + rentingRequest.getId());
               continue; // Skip this request
           }

           Double fineAmount = lateDays *
renting.getPricePerDay();

           // Update or create the Fine entity
           Fine fine = rentingRequest.getFine();
           if (fine == null) {
               fine = new Fine();
               fine.setDescription("Late return penalty
for " + lateDays + " days");
               fine.setAmount(fineAmount);
               fine.setUser(rentingRequest.getUser());
               fine.setRentingRequest(rentingRequest);
               fine.setIsPaid(false);
               rentingRequest.setFine(fine);
           } else {
               fine.setAmount(fineAmount); // Update the
existing fine
```

```
            }

            fineRepository.save(fine);
            System.out.println("Updated fine for
RentingRequest ID " + rentingRequest.getId() +
                    ": $" + fineAmount + " for " +
lateDays + " days late.");
        }
    }
}
    //Raghad
    // Method to mark a bike as returned
    public void markBikeAsReturned(Integer
rentingRequestId) {
        RentingRequest rentingRequest =
rentingRequestRepository.findRentingRequestById(rentingRe
questId);
          if(rentingRequest==null){
            throw new ApiException("Renting Request not
found");}
        Owner owner =
ownerRepository.findOwnerById(rentingRequest.getRenting()
.getOwner().getId());
          if(owner ==null){
              throw new ApiException("Just owner can mark
a bike as returned");
          }


        // Update the isReturned status
        rentingRequest.setIsReturned(true);
        rentingRequestRepository.save(rentingRequest);

        System.out.println("Bike marked as returned for
RentingRequest ID " + rentingRequestId);
    }


    public long getNumberOfFinesByUserId(Integer userId) {
        // Fetch the count of fines by user ID
        return fineRepository.countFinesByUserId(userId);
    }

//Raghad
```

```
//fine payment feature
   public void payFine(Integer fineId) {
       // Step 1: Fetch the fine
       Fine fine = fineRepository.findById(fineId)
               .orElseThrow(() -> new ApiException("Fine
not found"));

       // Step 2: Check if the fine is already paid
       if (Boolean.TRUE.equals(fine.getIsPaid())) {
           throw new ApiException("This fine has already
been paid");
       }

       // Step 3: Mark the fine as paid
       fine.setIsPaid(true);
       fineRepository.save(fine);

   }

   //Raghad
   public List<FineDTO> getUnpaidFinesByUserId(Integer
userId) {
       List<Fine> fines =
fineRepository.findUnpaidFinesByUserId(userId);
       List<FineDTO> fineDTOS=new ArrayList<>();

       for(Fine fine:fines){
           FineDTO fineDTO = new
FineDTO(fine.getDescription(),fine.getAmount(),fine.getIs
Paid());
           fineDTOS.add(fineDTO);
       }
       return fineDTOS;

   }
```

in OwnerService

```
//Raghad
   public List<OwnerDTO> getAllOwners() {
       // Step 1: Fetch all owners
       List<Owner> owners = ownerRepository.findAll();

       // Step 2: Map each owner to OwnerDTO
```

```
        List<OwnerDTO> ownerDTOList = new ArrayList<>();
        for (Owner owner : owners) {
            // Fetch motorcycles for the owner
            List<Motorcycle> motorcycles =
motorcycleRepository.findMotorcycleByOwnerId(owner.getId(
));

            // Map motorcycles to MotorcycleDTOs
            List<MotorcycleDTO> motorcycleDTOs =
motorcycles.stream().map(motorcycle -> new MotorcycleDTO(
                    motorcycle.getBrand(),
                    motorcycle.getModel(),
                    motorcycle.getYear(),
                    motorcycle.getPrice(),
                    motorcycle.getColor(),
                    motorcycle.getIsForSale(),
                    motorcycle.getIsAvailable(),
                    motorcycle.getHasOffer()
            )).collect(Collectors.toList());

            // Fetch courses for the owner
            List<Course> courses =
courseRepository.findCoursesByOwnerId(owner.getId());

            // Map courses to CourseDTOs
            List<CourseDTO> courseDTOs =
courses.stream().map(course -> new CourseDTO(
                    course.getName(),
                    course.getDescription(),
                    course.getPrice(),
                    course.getDuration()
            )).collect(Collectors.toList());

            // Map Owner to OwnerDTO
            OwnerDTO ownerDTO = new OwnerDTO(
                    owner.getName(),
                    owner.getEmail(),
                    owner.getPhoneNumber(),
                    owner.getAddress(),
                    motorcycleDTOs,
                    courseDTOs
            );

            ownerDTOList.add(ownerDTO);
```

```
        }

        return ownerDTOList;
    }
```

in RentingRequestService i do this methods(getAllRentingRequests - addRentingRequest-calculateTotalCost-updateRentingRequest)

in RentingRequestRepository

```
//Raghad
@Query("SELECT r FROM RentingRequest r WHERE r.endDate <
:currentDate")
List<RentingRequest>
findOverdueRentals(@Param("currentDate") LocalDate
currentDate);
```

in RentingRepository

```
  // Query to check availability of motorcycle for new
renting requests
//Raghad
    @Query("SELECT COUNT(r) > 0 FROM RentingRequest r " +
            "WHERE r.renting.motorcycleId = :motorcycleId
" +
            "AND (:startDate BETWEEN r.startDate AND
r.endDate " +
            "OR :endDate BETWEEN r.startDate AND
r.endDate)")
    boolean existsByMotorcycleAndDateRange(
            @Param("motorcycleId") Integer motorcycleId,
            @Param("startDate") LocalDate startDate,
            @Param("endDate") LocalDate endDate
    );

//Raghad
    // Query to check availability of motorcycle for
updates, excluding the current request
    @Query("SELECT COUNT(r) > 0 FROM RentingRequest r " +
            "WHERE r.renting.motorcycleId = :motorcycleId
" +
            "AND r.id <> :rentingRequestId " + // Exclude
the current renting request
            "AND (:startDate BETWEEN r.startDate AND
r.endDate " +
```

```
            "OR :endDate BETWEEN r.startDate AND
r.endDate)")
    boolean
existsByMotorcycleAndDateRangeExcludingRequest(
            @Param("motorcycleId") Integer motorcycleId,
            @Param("startDate") LocalDate startDate,
            @Param("endDate") LocalDate endDate,
            @Param("rentingRequestId") Integer
rentingRequestId
    );
```

in MotorcycleRepository

```
//Raghad
List<Motorcycle> findMotorcycleByOwnerId(Integer
owenrId);
```

in MaintenanceRequestRepository

```
//Raghad
@Query("SELECT m FROM MaintenanceRequest m WHERE
m.expert_name = :expertName AND m.pickupDate > :today")
List<MaintenanceRequest>
findUpcomingRequestsByExpert(String expertName,
LocalDate today);
```

in FineRepository

```
List<Fine> findFineByUserId(Integer userId);

@Query("SELECT COUNT(f) FROM Fine f WHERE f.user.id =
:userId")
long countFinesByUserId(@Param("userId") Integer userId);

@Query("SELECT f FROM Fine f WHERE f.user.id = :userId
AND f.isPaid = false")
List<Fine> findUnpaidFinesByUserId(@Param("userId")
Integer userId);
```

in CourseRepository

```
//Raghad
    @Query("SELECT c FROM Course c " +
            "WHERE (:minPrice IS NULL OR c.price >=
:minPrice) " +
```

```
            "AND (:maxPrice IS NULL OR c.price <=
:maxPrice) " +
            "AND (:minDuration IS NULL OR c.duration >=
:minDuration) " +
            "AND (:maxDuration IS NULL OR c.duration <=
:maxDuration) " )
    List<Course> filterCourses(
            @Param("minPrice") Double minPrice,
            @Param("maxPrice") Double maxPrice,
            @Param("minDuration") Integer minDuration,
            @Param("maxDuration") Integer maxDuration
    );
```

in BookingCourseRepository

```
//Raghad
    @Query("SELECT COUNT(b) > 0 FROM BookingCourse b " +
            "WHERE b.course.owner.id = :ownerId " +
            "AND (:startDate BETWEEN b.courseStartDate AND
b.courseEndDate " +
            "OR :endDate BETWEEN b.courseStartDate AND
b.courseEndDate)")
    boolean isTrainerUnavailable(
            @Param("ownerId") Integer ownerId,
            @Param("startDate") LocalDate startDate,
            @Param("endDate") LocalDate endDate
    );
```

classes

- AdminService/Model/Repository
- BookingCourseService i do some method in it
- CompanyService/Model/Repository
- CourseService i do some method in it
- EventService i do some method in it /Model/Repository
- FineService/Model/Repository
- RentingRequestService i do some method in it