Uygulama Team ve Player olmak üzere iki Entity'den oluşur. Team takımları, Player ise takımlarda oynayan oyuncuları temsil eder.

Bu iki Entity arasında bire çok ilişki kurulmuştur.

- Bir takımın birden fazla oyuncusu olabilir.
- Bir oyuncu yalnızca bir takıma aittir.

Team Sınıfı Özellikleri :

- Builder Patterni Kullanılmıstır.

Team sınıfına ait fieldlar :

  - id -> String tipinde boyutu en fazla 3 karakter olabilir
  - name -> String tipinde
  - country -> String tipinde

Team sınıfı içerisinde toDto() metodunu barındırır. Bu metod Builder ile bir TeamDto nesnesi oluşturup alanlarını team sınıfının içerisinden alır.

```
Burak Bulut
public TeamDto toDto() {
    return TeamDto.builder()
            .id(id)
            .name(name)
            .country(country)
            .playerList(playerList)
            .build();
}
```

Player Sınıfı Özellikleri :

- Builder Patterni Kullanılmıstır.

Player sınıfına ait fieldlar :

  - id -> Long tipinde , generationValue identitiy olarak ayarlandı.
  - fullName -> String tipinde
  - position-> String tipinde
  - teamId -> String tipinde , team tablosuna ait yabancıl anahtar.

Player sınıfı içerisinde toDto() metodunu barındırır. Bu metod Builder ile bir PlayerDto nesnesi oluşturup alanlarını player sınıfının içerisinden alır.

Repository Katmanı :

- Spring Data Jpa ile özelliği ile JpaRepository implementasyonu kullanılarak Data Access Layer oluşturulmuştur.

Team için :

```
Burak Bulut
@Repository
public interface TeamRepository extends JpaRepository<Team,String> {

}
```

Player için :

- Belirli bir teamId için oyuncuları getiren sorgu eklenmiştir.

```
Burak Bulut
@Repository
public interface PlayerRepository extends JpaRepository<Player,Long> {
    /**
     *
     * @param teamId String tipinde takimin id degerini alir
     * @return takimin idsine ait oyuncuları list tipinde dondurur.
     */
    Burak Bulut
    List<Player> findPlayersByTeamId(String teamId);
}
```

Service Katmanı :

- Service katmanı interface ve interfaceleri implemente eden impl sınıfları olacak şekilde uygulanmıştır.
- Service katmanında api'ların istek attığı metodlar dto nesnesi döndürür.

Team Service Özellikleri :

- *findAll()* -> Repositoryden gelen team listesi TeamDtoList sınıfına gönderilir. Return type : TeamDtoList.
- *findById(String id)* -> Repositoryden gelen team nesnesi, team sınıfının içerisinde bulunan toDto() metodu ile dto nesnesine dönüştürülür. Idye ait takım bulunamaz ise exception verir. Return type : TeamDto, Exception message : Takım Bulunamadı
- *save(TeamDto dto)* -> dto, TeamDto sınıfının toEntity() metoduyla Team nesnesine dönüştürülüp save işlemi gerçekleştirilir ardından save metodunun döndürdüğü Team nesnesi tekrar dtoya dönüştürülür. Return type : TeamDto.
- *update(String id, TeamDto dto)* -> Önce Repositoryden güncellenecek team nesnesi aranır. Bulunamazsa exception verir. Bulunursa dto ile gönderilen alanlar ternary if ile kontrol edilir ardından save işlemi yapılır ve dönen team nesnesi dto ya dönüştürülür. Return type : TeamDto. Exception Message : Takım Bulunamadı.
- *delete(String id)* -> Gönderilen id'ye ait team var mi kontrol edilir varsa silme işlemi yapılır yoksa exception verir. Return type : Void Exception Message : Takım Bulunamadı.
- *getById(String id)* -> Girilen id'ye ait team nesnesini döndürür. Return Type : Team.

Player Service Özellikleri :

- *findAll()* -> Repositoryden gelen player listesi PlayerDtoList sınıfına gönderilir. Return type : PlayerDtoList.
- *findPlayersByTeamId(String teamId)* -> Girilen teamId'ye göre gelen player listesi PlayerDtoList sınıfına gönderilir. Return Type: PlayerDtoList
- *findById(Long id)* -> Repositoryden gelen player nesnesi, player sınıfının içerisinde bulunan toDto() metodu ile dto nesnesine dönüştürülür. Idye ait player bulunamaz ise exception verir. Return type : PlayerDto, Exception message : Oyuncu Bulunamadı
- *save(PlayerDto dto)* -> Önce dto'nun içerisinde bulunan team nesnesinin id'sine ait team varmı diye kontrol edilir yoksa exception verir. dto, PlayerDto sınıfının toEntity() metoduyla Player nesnesine dönüştürülüp save işlemi gerçekleştirilir ardından save metodunun döndürdüğü Player nesnesi tekrar dtoya dönüştürülür. Return type : PlayerDto.
- *update(Long id, PlayerDto dto)* -> Önce Repositoryden güncellenecek Player nesnesi aranır. Bulunamazsa exception verir. Bulunursa dto ile gönderilen alanlar ternary if ile kontrol edilir ardından save işlemi yapılır ve dönen player nesnesi dto ya dönüştürülür. Return type : PlayerDto. Exception Message : Player Bulunamadı.
- *delete(Long id)* -> Gönderilen id'ye ait player var mi kontrol edilir varsa silme işlemi yapılır yoksa exception verir. Return type : Void Exception Message : Player Bulunamadı.

TeamServiceImpl Sınıfı :

```java
@Service
@RequiredArgsConstructor
public class TeamServiceImpl implements TeamService {

    private final TeamRepository repository;

    /** Tum takimlari iceren listeyi dtoList seklinde dondurur. ...*/
    Burak Bulut
    @Override
    public TeamDtoList findAll() { return new TeamDtoList(repository.findAll()); }

    /** Girilen id'ye gore takimi dto seklinde dondurur. ...*/
    Burak Bulut
    @Override
    public TeamDto findById(String id) {
        return repository.findById(id).orElseThrow(()-> new NotFoundException("Takim Bulunamadi")).toDto();
    }

    /** Gonderilen Takimi veritabanina kaydeder daha sonra entityi dtoya dondurup return eder. ...*/
    Burak Bulut
    @Override
    public TeamDto save(TeamDto dto) throws Exception {
        if (repository.findById(dto.getId()).isPresent()){
            throw new Exception("Bu idye ait takim bulunuyor.");
        }
        return repository.save(dto.toEntity()).toDto();
    }

    /** Girilen id'ye gore takimi bulup dto nesnesinde doldurulan alanlar icin guncelleme islemini yapar. ...*/
    Burak Bulut
    @Override
    public TeamDto update(String id, TeamDto dto) {
        Optional<Team> updatedTeam = repository.findById(id);
        if (updatedTeam.isPresent()){
            Team team = updatedTeam.get().builder()
                    .id(updatedTeam.get().getId())
                    .name(dto.getName()!=null ? dto.getName() : updatedTeam.get().getName())
                    .country(dto.getCountry()!=null ? dto.getCountry() : updatedTeam.get().getCountry())
                    .playerList(updatedTeam.get().getPlayerList())
                    .build();
            return repository.save(team).toDto();
        }
        throw new NotFoundException("Guncellenecek Takim Bulunamadi");
    }

    /** Gonderilen id numarasina ait takimi siler. ...*/
    Burak Bulut
    @Override
    public Void delete(String id) {
        if(repository.findById(id).isPresent()){
            repository.deleteById(id);
            return null;
        }
        throw new NotFoundException("Silinecek Takim Bulunamadi");
    }

    /** Gonderilen id ye gore takimi getirir. ...*/
    Burak Bulut
    @Override
    public Team getById(String id) {
        return repository.findById(id).orElseThrow(()-> new NotFoundException("Takim Bulunamadi"));
    }
}
```

PlayerServiceImpl :

```java
/** PlayerService implamantasyonu */
Burak Bulut
@Service
@RequiredArgsConstructor
public class PlayerServiceImpl implements PlayerService {
    private final PlayerRepository repository;
    private final TeamService teamService;

    /** Tum oyuncuları iceren listeyi dtoList seklinde dondurur. ...*/
    Burak Bulut
    @Override
    public PlayerDtoList findAll() { return new PlayerDtoList(repository.findAll()); }

    /** Girilen id'ye gore oyuncuyu dto seklinde dondurur. ...*/
    Burak Bulut
    @Override
    public PlayerDto findById(Long id) {
        return repository.findById(id).orElseThrow(() -> new NotFoundException("Oyuncu Bulunamadi")).toDto();
    }

    /** Takimlarda oynayan oyuncuları dtolist seklinde dondurur. ...*/
    Burak Bulut
    @Override
    public PlayerDtoList findPlayersByTeamId(String teamId) {
        return new PlayerDtoList(repository.findPlayersByTeamId(teamId));
    }

    /** Once gonderilen oyuncu nesnesinin icerisindeki teamId ye ait bir takim olup olmadigi kontrol edilir. ...*/
    Burak Bulut
    @Override
    public PlayerDto save(PlayerDto dto) {
        Team team = teamService.getById(dto.getTeam().getId());
        dto.setTeam(team);
        return repository.save(dto.toEntity()).toDto();
    }

    /** Girilen id'ye gore oyuncuyu bulup dto nesnesinde doldurulan alanlar icin guncelleme islemini yapar. ...*/
    Burak Bulut
    @Override
    public PlayerDto update(Long id, PlayerDto dto) {
        Optional<Player> updatedPlayer = repository.findById(id);
        if (updatedPlayer.isPresent()) {
            Player player = updatedPlayer.get().builder()
                    .id(updatedPlayer.get().getId())
                    .fullName(dto.getFullName() != null ? dto.getFullName() : updatedPlayer.get().getFullName())
                    .position(dto.getPosition() != null ? dto.getPosition() : updatedPlayer.get().getPosition())
                    .team(dto.getTeam() != null ? teamService.getById(dto.getTeam().getId()) : updatedPlayer.get().getTeam())
                    .build();
            return repository.save(player).toDto();
        }
        throw new NotFoundException("Guncellencek Kullanici Bulunamadi ");
    }

    /** Gonderilen id numarasina ait takimi siler. ...*/
    Burak Bulut
    @Override
    public Void delete(Long id) {
        if (repository.findById(id).isPresent()) {
            repository.deleteById(id);
            return null;
        }
        throw new NotFoundException("Silinecek Kullanici Bulunamadi");
    }
}
```

Dto Nesneleri :

Team için Dto :

```java
/** Team sinifinin dto objesini temsil eder. */
Burak Bulut
@Data
@Builder
public class TeamDto {
    private String id;
    private String name;
    private String country;
    @JsonIgnore
    private List<Player> playerList;

    /** Dto objesini Entity objesine cevirir. ...*/
    Burak Bulut
    public Team toEntity(){
        return Team.builder()
                .id(id)
                .name(name)
                .country(country)
                .playerList(playerList)
                .build();
    }
}
```

```java
/** Dto nesnelerini list seklinde donduren sinifdir. */
Burak Bulut
@Data
public class TeamDtoList {
    private List<TeamDto> teamDtoList;

    Burak Bulut
    public TeamDtoList(List<Team> teamList) {
        this.teamDtoList = teamList.stream() Stream<Team>
                .map(Team::toDto) Stream<TeamDto>
                .collect(Collectors.toList());
        getTeamDtoList();
    }
}
```

Player için Dto :

```java
/** Player sinifinin dto objesini temsil eder. */
Burak Bulut
@Data
@Builder
public class PlayerDto {
    private Long id;
    private String fullName;
    private String position;
    @JsonIgnore
    private Team team;

    /** Dto objesini Entity objesine cevirir. ...*/
    Burak Bulut
    public Player toEntity(){
        return Player.builder()
                .fullName(fullName)
                .team(team)
                .position(position)
                .build();
    }

}
```

```java
/** Dto nesnelerini list seklinde donduren sinifdir. */
Burak Bulut
@Data
public class PlayerDtoList {
    private List<PlayerDto> playerDtoList;

    Burak Bulut
    public PlayerDtoList(List<Player> playerList) {
        this.playerDtoList = playerList.stream() Stream<Player>
                .map(Player::toDto) Stream<PlayerDto>
                .collect(Collectors.toList());
        getPlayerDtoList();
    }
}
```

Response Nesneleri :

Team için Response :

```java
/** Team sinifina ait response nesnesi. */
Burak Bulut
@Data
@Builder
public class TeamResponse {
    private String id;
    private String name;
    private String country;
    private List<PlayerDto> playerList;

    /** Gelen dto nesnesini response nesnesine donusturur. ...*/
    Burak Bulut
    public static TeamResponse toResponse(TeamDto dto){
        return TeamResponse.builder()
                .id(dto.getId())
                .name(dto.getName())
                .country(dto.getCountry())
                .playerList(new PlayerDtoList(dto.getPlayerList()).getPlayerDtoList())
                .build();
    }
}
```

```java
/** Response nesnelerini liste seklinde dondurur. */
Burak Bulut
@Data
public class TeamResponseList {
    private List<TeamResponse> teamResponses;
    Burak Bulut
    public TeamResponseList(List<TeamDto> teamDtoList) {
        this.teamResponses = teamDtoList
                .stream() Stream<TeamDto>
                .map(TeamResponse::toResponse) Stream<TeamResponse>
                .collect(Collectors.toList());
        getTeamResponses();
    }
}
```

Player için Response :

```java
/** Player sinifina ait response nesnesi. */
Burak Bulut
@Data
@Builder
public class PlayerResponse {
    private Long id;
    private String fullName;
    private TeamDto team;
    private String position;

    /** Gelen dto nesnesini response nesnesine donusturur. ...*/
    Burak Bulut
    public static PlayerResponse toResponse(PlayerDto dto){
        return PlayerResponse.builder()
                .id(dto.getId())
                .fullName(dto.getFullName())
                .team(dto.getTeam().toDto())
                .position(dto.getPosition())
                .build();
    }

}
/** Response nesnelerini liste seklinde dondurur. */
Burak Bulut
@Data
public class PlayerResponseList {
    private List<PlayerResponse> playerResponseList;

    Burak Bulut
    public PlayerResponseList(List<PlayerDto> playerDtoList) {
        this.playerResponseList = playerDtoList
                .stream() Stream<PlayerDto>
                .map(PlayerResponse::toResponse) Stream<PlayerResponse>
                .collect(Collectors.toList());
        getPlayerResponseList();
    }
}
```

Request Nesneleri :

Team için :

```java
/**
 * Team'a ait request nesnesi.
 */
Burak Bulut *
@Data
public class TeamRequest {
    private String id;
    private String name;
    private String country;

    /** Requesti dtoya dönüştürür. ...*/
    Burak Bulut
    public TeamDto toDto(){
        return TeamDto.builder()
                .id(id)
                .name(name)
                .country(country)
                .playerList(new ArrayList<>())
                .build();
    }
}
```

Player için :

```java
/** Playera ait request nesnesi. */
Burak Bulut
@Data
public class PlayerRequest {
    private String fullName;
    private String position;
    private String teamId;

    /** Requesti dtoya dönüştürür. ...*/
    Burak Bulut
    public PlayerDto toDto(){
        /// Update isleminde takim degistirilmediğinde daha takımı null yapan sorun giderildi.
        if (teamId == null){
            return PlayerDto.builder()
                    .fullName(fullName)
                    .position(position)
                    .build();
        }
        return PlayerDto.builder()
                .fullName(fullName)
                .position(position)
                .team(Team.builder().id(teamId).build())
                .build();
    }
}
```

Controller Katmanı :

- ResponseEntity<> sınıfı kullanıldı.
- ResponseEntity nin ok ve badRequest metodları üzerinden senaryolar işlendi.
- Controller Katmanı için BaseResponse<T> isimli bir generic sınıf üretildi. Bu sınıf içerisinde başarılı veya başarısız durumlar için status ve message gibi alanlar bulundurur.
- http tarafına gönderilen post ve put requestleri için her bir entitye ait request sınıfları üretildi.

BaseResponse :

```java
/** Controllarlar icin temel response sinifidir. ...*/
Burak Bulut
@Data
@Builder
public class BaseResponse<T> {
    private Integer status;
    private String message;
    private T response;

    /** Basarili bir request sonucunda request ile ilgili status message ve response ile donen datayi icerir. ...*/
    Burak Bulut
    public static <T> BaseResponse<T> success(T data, String message) {
        return BaseResponse.<T>builder()
                .status(200)
                .message(message)
                .response(data)
                .build();
    }

    /** Basarisiz bir request sonucunda status ve message bilgilerini icerir. ...*/
    Burak Bulut
    public static <T> BaseResponse<T> error(String message) {
        return BaseResponse.<T>builder()
                .status(400)
                .message(message)
                .build();
    }
}
```

Controller Nesneleri :

Team için :

```java
/** Team sinifinin api nesnesidir. ...*/
Burak Bulut *
@RestController
@RequestMapping("teams")
@RequiredArgsConstructor
public class TeamController {
    private final TeamService service;

    Burak Bulut *
    @GetMapping
    private ResponseEntity<BaseResponse<TeamResponseList>> getTeams() {
        try {
            return ResponseEntity.ok(BaseResponse.success
                    (new TeamResponseList(service.findAll().getTeamDtoList()), message: "Takimlar Basariyla Getirildi"));
        } catch (Exception e) {
            return ResponseEntity.badRequest().body(BaseResponse.error(e.getMessage()));
        }
    }

    Burak Bulut
    @GetMapping("{id}")
    private ResponseEntity<BaseResponse<TeamResponse>> getTeamById(@PathVariable String id) {
        try {
            return ResponseEntity.ok(BaseResponse.success(TeamResponse.toResponse(service.findById(id)), message: "Takim Bulundu"));
        } catch (NotFoundException e) {
            return ResponseEntity.badRequest().body(BaseResponse.error(e.getMessage()));
        }
    }

    Burak Bulut
    @PostMapping
    private ResponseEntity<BaseResponse<TeamResponse>> save(@RequestBody TeamRequest request) {
        try {
            return ResponseEntity.ok(BaseResponse.success(TeamResponse.toResponse(service.save(request.toDto())), message: "Kayit Basarili"));
        } catch (Exception e) {
            return ResponseEntity.badRequest().body(BaseResponse.error(e.getMessage()));
        }
    }

    Burak Bulut
    @PutMapping("{id}")
    private ResponseEntity<BaseResponse<TeamResponse>> update(@PathVariable String id, @RequestBody TeamRequest request) {
        try {
            return ResponseEntity.ok(BaseResponse.success(TeamResponse.toResponse(service.update(id, request.toDto())), message: "Update Basarili"))
        } catch (NotFoundException e) {
            return ResponseEntity.badRequest().body(BaseResponse.error(e.getMessage()));
        }
    }

    Burak Bulut
    @DeleteMapping("{id}")
    private ResponseEntity<BaseResponse<Void>> delete(@PathVariable String id) {
        try {
            return ResponseEntity.ok(BaseResponse.success(service.delete(id), message: "Silme İslemi Basarili"));
        } catch (NotFoundException e) {
            return ResponseEntity.badRequest().body(BaseResponse.error(e.getMessage()));
        }
    }
}
```

Player için :

```java
/** Player sinifinin api nesnesidir. ...*/
Burak Bulut *
@RestController
@RequestMapping("players")
@RequiredArgsConstructor
public class PlayerController {
    private final PlayerService service;

    Burak Bulut *
    @GetMapping
    private ResponseEntity<BaseResponse<PlayerResponseList>> getPlayers() {
        try {
            return ResponseEntity.ok(BaseResponse.success
                    (new PlayerResponseList(service.findAll().getPlayerDtoList()), message: "Oyuncular Basariyla Getirildi"));
        } catch (NotFoundException e) {
            return ResponseEntity.badRequest().body(BaseResponse.error(e.getMessage()));
        }
    }

    Burak Bulut *
    @GetMapping("/teamId/{id}")
    private ResponseEntity<BaseResponse<PlayerResponseList>> getPlayersByTeamId(@PathVariable String id) {
        try {
            return ResponseEntity.ok(BaseResponse.success
                    (new PlayerResponseList(service.findPlayersByTeamId(id).getPlayerDtoList()), message: "Oyuncular Basariyla Getirildi"));
        } catch (NotFoundException e) {
            return ResponseEntity.badRequest().body(BaseResponse.error(e.getMessage()));
        }
    }

    Burak Bulut *
    @GetMapping("/playerId/{id}")
    private ResponseEntity<BaseResponse<PlayerResponse>> getPlayerById(@PathVariable String id) {
        try {
            return ResponseEntity.ok(BaseResponse.success
                    (PlayerResponse.toResponse(service.findById(Long.parseLong(id))), message: "Oyuncu Bulundu"));
        } catch (NotFoundException e) {
            return ResponseEntity.badRequest().body(BaseResponse.error(e.getMessage()));
        }
    }

    Burak Bulut
    @PostMapping
    private ResponseEntity<BaseResponse<PlayerResponse>> save(@RequestBody PlayerRequest request) {
        try {
            return ResponseEntity.ok(BaseResponse.success(PlayerResponse.toResponse(service.save(request.toDto())), message: "Kayit Basarili"));
        } catch (NotFoundException e) {
            return ResponseEntity.badRequest().body(BaseResponse.error(e.getMessage()));
        }
    }

    Burak Bulut *
    @PutMapping("{id}")
    private ResponseEntity<BaseResponse<PlayerResponse>> update(@PathVariable(name = "id") String id, @RequestBody PlayerRequest request) {
        try {
            return ResponseEntity.ok(BaseResponse.success
                    (PlayerResponse.toResponse(service.update(Long.parseLong(id), request.toDto())), message: "Update Basarili"));
        } catch (NotFoundException e) {
            return ResponseEntity.badRequest().body(BaseResponse.error(e.getMessage()));
        }
    }

    Burak Bulut
    @DeleteMapping("{id}")
    private ResponseEntity<BaseResponse<Void>> delete(@PathVariable String id) {
        try {
            return ResponseEntity.ok(BaseResponse.success(service.delete(Long.parseLong(id)), message: "Silme İslemi Basarili"));
        } catch (NotFoundException e) {
            return ResponseEntity.badRequest().body(BaseResponse.error(e.getMessage()));
        }
    }
}
```

Postman Sorguları :