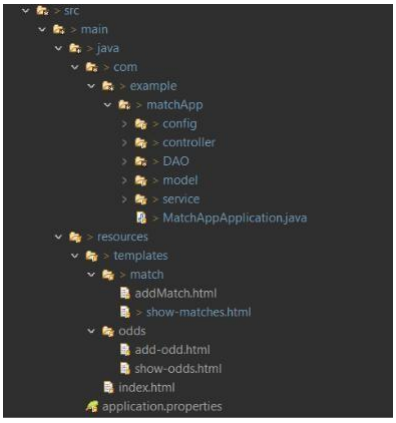
The Match Application

Trampas Eleftherios

To Project.

Οσκοπός του project αυτού, είναι η διαχείριση αγώνων και των αποδόσεων, κάθε αγώνα. Σε αυτό συμπεριλαμβάνεται, η προσθήκη ενός αγώνα και των αποδόσεων του, η επεξεργασία αυτού και της/των απόδοσης/ων και τέλος η διαγραφή τους, αν είναι επιθυμητή. Η υλοποίηση του project έγινε σε java spring boot στο Eclipse IDE. Παρακάτω υπάρχει μια σειρά γεγονότων για το πως προχώρησε η διαδικασία υλοποίησης.

Το συνολικό project σε πακέτα :



Spring Boot.

Η Spring boot είναι ένα framework της Spring που είναι ένα framework της JAVA. Χρησιμοποιείται για την ανάπτυξη εφαρμογών κάνοντας όλη την διαδικασία πιο εύκολη και κατανοητή. Για παράδειγμα, έχει αυτόνομο web server (πχ Tomcat) με αποτέλεσμα να μην χρειάζεται η δημιουργία κάποιου εξωτερικού. Ακόμη ένα από τα πλεονεκτήματα της, είναι ότι η σύνδεση με την βάση γίνεται αυτόματα στα **dependencies** του **.xml** αρχείου, με τη μόνη παρεμβολή του προγραμματιστή να είναι, η συμπλήρωση του string στο application.properties το οποίο έχει την διεύθυνση που «ακούει» η βάση και γενικά πληροφορίες όπως, όνομα χρήστη, κωδικό πρόσβασης, την θύρα κλπ.

Τέλος, στην Spring boot, υπάρχουν και τα annotation τα οποία κάνουν ακόμη πιο απλοϊκό τον κώδικα μας. Παρακάτω θα δείτε ότι πάνω από μία κλάση ή μέθοδο ή πεδίο υπάρχει ένα σύμβολο «@» που ακολουθείτε από μια λέξη. Αυτό συμβαίνει ώστε να καταλάβει η spring boot το τι συμβαίνει στον κώδικα μας και να προχωρήσει με τις κατάλληλες ενέργειες. Θα εξηγηθούν και παρακάτω.

Η **υλοποίηση.**

Η αρχιτεκτονική είναι model-view-controller. Το μοντέλο μου επικοινωνεί με το view, δηλαδή ότι αφορά front-end (HTML, CSS, JS κλπ.) και οι controllers επικοινωνούν με τον χρήστη και διαχειρίζονται τις ενέργειες του.

Το **μοντέλο** οντοτήτων περιλαμβάνει έναν αγώνα και μια απόδοση αγώνα, καθώς και το άθλημα το οποίο λαμβάνει χώρα. Ακολουθούν στιγμιότυπα με τα παραπάνω.

```java
@Entity
@Table(name="match")
public class Match {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="match_id")
    private int id;
    @Column(name="description")
    private String description;
    @Column(name="match_date")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate match_date;
    @Column(name="match_time")
    private String match_time;
    @Column(name="team_a")
    private String team_a;
    @Column(name="team_b")
    private String team_b;
    @Enumerated(EnumType.ORDINAL)
    @Column(name = "sport")
    private Sport sport;
    @OneToMany(mappedBy = "match", cascade = CascadeType.ALL)
    private List<MatchOdd> odds = new ArrayList<>();
```

Εικόνα 1. Match Entity

```java
@Entity
@Table(name="matchOdds")
public class MatchOdd {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private int id;
    @ManyToOne
    @JoinColumn(name = "match_id", nullable = false)
    private Match match;
    @Column(name="specifier")
    private String specifier;
    @Column(name="odd")
    private double odd;
```

Εικόνα 2. Match Odd Entity

Εικόνα 3. Sport Enum

Οι εικόνες αναπαριστούν το μοντέλο του project. Έχουμε την οντότητα Match με τα αντίστοιχα γνωρίσματα id, description, match date, match time, team a, team b. Επιπλέον στην οντότητα έχουμε το πεδίο sport ως enumerated όπου κρατάμε μία αξία για να προσδιορίσουμε το άθλημα αν είναι FOOTBALL or BASKETBALL. Το annotation id δείχνει ότι το πεδίο είναι id (Primary key) και το από κάτω ακριβώς χρησιμοποιείται για να δηλώσουμε ότι είναι αυξανόμενο-παραγόμενο αυτόματα (auto-increment-generated). Τα υπόλοιπα αφορούν τις στήλες και το όνομα που θα έχουν στην βάση, δηλαδή λέμε ότι το τάδε πεδίο θα είναι στήλη στον πίνακα με το τάδε όνομα . Τέλος, δημιουργούμε την συσχέτιση Match – Match Odd. Ο αγώνας έχει απόδοση. Ένας αγώνας έχει από μία ως πολλές αποδόσεις, ενώ μία απόδοση απευθύνεται σε έναν αγώνα. Συμπεραίνουμε ότι η συσχέτιση από αγώνα προς απόδοση είναι ένα προς πολλά και δηλώνεται ως @ One To Many στον αγώνα και ως @ Many To One στην απόδοση.

Όσον αφορά τα @ One To Many, @ Many To One, τα βάζουμε για δηλώσουμε την παραπάνω συσχέτιση. Στην οντότητα Match Odd, θέλουμε ως ξένο κλειδί το συνθηματικό του αγώνα, άρα έχουμε το πεδίο αγώνα μέσα σε αυτή αλλά και την γραμμή Join Column ώστε αυτό να μπει σαν στήλη στον πίνακα με τις αποδόσεις. Όμως το κάνουμε με αυτόν τον τρόπο επειδή υπάρχει συσχέτιση και γι' αυτό δεν βάζουμε μία απλή στήλη. Επιπλέον, στην οντότητα αγώνας κάνουμε map τον πίνακα αυτόν με τον πίνακα της συσχέτισης, λέγοντας το όνομα του πεδίου.

Ομοίως, η οντότητα Match Odd έχει τα γνωρίσματα της, όπου είναι τα id, specifier και odd, καθώς και την οντότητα match, όπου λειτουργεί αναλόγως όπως το odds στην οντότητα Match. Χρησιμοποιώ getters and setters για να μπορώ να επεξεργαστώ πληροφορίες των οντοτήτων.

Στο επόμενο στάδιο περνάμε στην σύνδεση με την βάση δεδομένων στην προκειμένη PostgreSQL, η οποία γίνεται στο application.properties(Εικόνα 4). Επιπλέον, στην σύνδεση της βάσης με το μοντέλο μας ρόλο σημαντικό παίζει το JPA Repository, το οποίο παρέχει έτοιμες εντολές αναζήτησης στην βάση.

```
spring.datasource.url=jdbc:postgresql://localhost:5432/matchDB
spring.datasource.username=postgres
spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration
```

Εικόνα 4. application. Properties

Προχωράω στη δημιουργία του πακέτου **DAO** ή **Repository** υπεύθυνο για επικοινωνία με την βάση(Εικόνα 5).

```
@Repository
public interface MatchDAO extends JpaRepository<Match, Integer> {

}
```

Εικόνα 5. Match DAO

Ομοίως και για το repository των αποδόσεων. Κενά, καθώς μπορώ να χρησιμοποιήσω τις έτοιμες μεθόδους της βιβλιοθήκης JPA όπου και αυτές χρειάζομαι, και ορισμένα με @Repository.

Περνάμε στη δημιουργία του **Service**, το οποίο είναι υπεύθυνο για την λειτουργικότητα του συστήματος. Η αρχιτεκτονική που χρησιμοποιείται είναι ότι έχουμε μια κλάση Interface και μια κανονική, για το ίδιο service (πχ. Match Service, Match Service Implementation). Ο σκοπός που γίνεται αυτό, είναι για να διαχωριστεί η ιδέα του business κομματιού από την κύρια λειτουργία του προϊόντος, με αποτέλεσμα να υπάρχει καλό readability του κώδικα, εύκολη επεξεργασία χωρίς να επηρεαστούν άλλες υλοποιήσεις που χρησιμοποιούν το service.

Στο συγκεκριμένο project, χρησιμοποίησα ένα service για τον αγώνα όπως φαίνεται παρακάτω.

```
import java.util.List;

public interface MatchService {
    void saveMatch(Match match);
    List<Match> getMatches();
    void deleteMatch(int id);
    List<MatchOdd> getMatchOdds(int id);
    void addOdd(MatchOdd odd, int matchId);
    void deleteOdd(int id);
    MatchOdd getMatchOdd(int oddId);
    Match getMatch(int id);
}
```

Εικόνα 6. Match Service (Interface).

```java
@Service
public class MatchServiceImpl implements MatchService {

    @Autowired
    private MatchDAO matchDAO;
    @Autowired
    private MatchOddDAO matchOddsDAO;

    @Override
    public void saveMatch(Match match) {
        matchDAO.save(match);
    }

    @Override
    public List<Match> getMatches() {
        return matchDAO.findAll();
    }

    @Override
    public void deleteMatch(int id) {
        matchDAO.deleteById(id);
    }

@Override
public Match getMatch(int id) {
    Optional<Match> match = matchDAO.findById(id);
    if(match.isPresent()) {
        return match.get();
    }else {
        throw new NullPointerException("Cannot find match with id = " + id);
    }
}
```

```java
@Override
public List<MatchOdd> getMatchOdds(int id) {
    Optional<Match> match = matchDAO.findById(id);
    if(match.isPresent()) {
        return match.get().getOdds();
    }else {
        throw new NullPointerException("Cannot find match odd with id = " + id);
    }

}

@Override
public MatchOdd getMatchOdd(int id) {
    Optional<MatchOdd> matchOdd = matchOddsDAO.findById(id);
    if(matchOdd.isPresent()) {
        return matchOdd.get();
    }else {
        throw new NullPointerException("Cannot find match odd with id = " + id);
    }
}

@Override
public void addOdd(MatchOdd odd, int matchId) {
    Match match = matchDAO.findById(matchId).get();
    odd.setMatch(match);
    matchOddsDAO.save(odd);
    return;
}

@Override
public void deleteOdd(int id) {
    matchOddsDAO.deleteById(id);
}
```

Εικόνα 7. Match Service Implementation

Το service χρησιμοποιεί το @ Service για να καταλάβει η spring ότι είναι service(μπαίνει σε ένα από τα δύο). Επικοινωνεί με την βάση, γι' αυτό χρησιμοποιούμε αντικείμενα από τα αντίστοιχα repositories που θέλουμε, βάζοντας @ Autowired. Το συγκεκριμένο annotation μία από τις χρήσεις που έχει, είναι η δημιουργία αντικειμένου αυτόματα χωρίς να δημιουργούμε εμείς ένα από την αρχή, γνωστό ως και dependency injection. Το spring εν τέλει αναγκάζεται να βρει πιο είναι αυτό το αντικείμενο που έχω δηλώσει ως @ Autowired και επιστρέφει την υλοποίηση του στην προκειμένη. Τέλος, όπου υπάρχει @ Override, δηλώνουμε πως η συγκεκριμένη μέθοδος που το έχει επικαλύπτει την ίδια μέθοδο όπου χρησιμοποιείται σε κάποιο interface (ή υπερκλάση). Χρησιμεύει και στο readability.

Έχουμε κάποιες **μεθόδους**. Ας τις εξηγήσουμε :

- **Save Match()** : Δέχεται έναν αγώνα και τον κάνει save αυτόματα το JPA στην βάση. Αυτό γίνεται με το να καλέσουμε την μέθοδο save(Match) με χρήση του αντικειμένου Match DAO.
- **Get Matches(), Delete Match(), Delete Odd()** : Ομοίως, σαν το Save Match(), με έτοιμη μέθοδο υλοποιούνται και αυτές.
- **Get Match(), Get Match Odds(), Get Match Odd()** : Και οι τρεις χρησιμοποιούν Optional<> για έλεγχο του αντικειμένου με .isPresent(). Η ιδέα, είναι η δημιουργία του αντικειμένου «αγώνα», για να πάρω το τρέχον id που θα χρειαστώ μελλοντικά και να επιστρέψω ολόκληρο το αντικείμενο. Αλλιώς, exception.
- **Add Odd()** : Παίρνω το τρέχον συνθηματικό του αγώνα που ζητάω για να προσθέσω απόδοση και περνάω αυτήν/ες στον αγώνα. Τέλος την αποθηκεύω με έτοιμη μέθοδο του JPA.

Οι παραπάνω μέθοδοι έχουν οριστεί στο Interface Match Service.

Συνεχίζω με τον **Controller** (Εικόνα 8). Συγκεκριμένα εδώ έχουμε έναν ο οποίος είναι υπεύθυνος για την σύνδεση των HTTP αιτήσεων (ή τα λεγόμενα requests) με τις λειτουργίες του συστήματος (Service). Ορίζουμε με @ Controller ώστε να έχουμε view.

```
@Controller
public class MatchController {
    @Autowired
    private MatchService matchService;

    @RequestMapping("/")
    public String home() {
        return "index";
    }

    @RequestMapping("/add")
    public String addMatch(Model model) {
        model.addAttribute("match", new Match());
        return "match/add-match";
    }

    @RequestMapping("/updateMatch")
    public String addMatch(Model model, @RequestParam("matchId") int matchId)
        model.addAttribute("match", matchService.getMatch(matchId));
        return "match/add-match";
    }

    @RequestMapping("/showMatches")
    public String showMatches(Model model) {

        model.addAttribute("matches", matchService.getMatches());
        return "match/show-matches";
    }

    @RequestMapping("/deleteMatch")
    public String deleteMatch(@RequestParam("matchId") int matchId) {

        matchService.deleteMatch(matchId);
        return "redirect:/showMatches";
```

```
@RequestMapping("/showOdds")
public String showOdds(Model model, @RequestParam("matchId") int matchId) {
    model.addAttribute("odds", matchService.getMatchOdds(matchId));
    model.addAttribute("matchId", matchId);
    return "odds/show-odds";
}

@RequestMapping("/addOdd")
public String addOdd(Model model, @RequestParam("matchId") int matchId) {
    model.addAttribute("odd", new MatchOdd());
    model.addAttribute("matchId", matchId);
    return "odds/add-odd";
}

@PostMapping("/addOdd")
public String addOdd(@RequestParam("matchId") int matchId, @ModelAttribute MatchOdd odd) {
    matchService.addOdd(odd, matchId);
    return "redirect:/showOdds?matchId="+matchId;
}

@RequestMapping("/save")
public String createMatch(@ModelAttribute("match") Match match, Model model) {
    matchService.saveMatch(match);
    return "redirect:/showMatches";
}

@RequestMapping("/updateOdd")
public String updateOdd(Model model, @RequestParam("matchId") int matchId, @RequestParam("oddId") int oddId) {
    model.addAttribute("odd", matchService.getMatchOdd(oddId));
    model.addAttribute("matchId", matchId);

    return "odds/add-odd";

@RequestMapping("/deleteOdd")
public String deleteOdd(@RequestParam("matchId") int matchId, @RequestParam("oddId") int oddId) {
    matchService.deleteOdd(oddId);
    return "redirect:/showOdds?matchId="+matchId;
```

Εικόνα 8. Controller

Η γενική ιδέα είναι ότι δημιουργώντας ένα κουμπί ή ένα action σε HTML, αυτό το οποίο θα αλληλοεπιδράσει ο χρήστης, τότε κάνει αίτηση σε εμάς ότι κάτι θέλει. Έτσι δημιουργώντας κάτι τέτοιο, τότε στο HTML (όπως θα εξηγηθεί πιο κάτω) έχουμε βάλει ότι όταν γίνει μια ενέργεια Χ από τον χρήστη, τότε η ενέργεια αυτή επικοινωνεί με βάση αυτό που υπάρχει μέσα στην **λεζάντα** @Request Mapping(«ΛΕΖΑΝΤΑ»). Για παράδειγμα αν έχω ένα κουμπί και θέλω να επικοινωνήσω με συγκεκριμένη μέθοδο του controller τότε γράφω στο κουμπί σε HTML, την **λεζάντα**, ώστε να πάει στον Controller με την ίδια **λεζάντα** και να εκτελέσει την λειτουργία.

Όταν έχω σαν όρισμα **«Model»**, το κάνω ώστε να πάρω από το html ότι έχω δηλώσει ότι απευθύνομαι. Δηλαδή, αν έχω δηλώσει στο HTML object = match, τότε το **μοντέλο** μου περιμένει να δει έναν αγώνα που θα του δώσω ως model.addAtribute() και θα γίνει η σύνδεση.

Οι μέθοδοι :

- **Home()** : Επιστρέφει ένα αρχείο “index.html”. Φορτώνει την αρχική σελίδα.
- **Add Match()** : Πριν συμπληρώσουμε την φόρμα προσθήκης αγώνα, δημιουργούμε ένα αντικείμενο αγώνα, ώστε να καταχωρήσουμε τις πληροφορίες του. Επιστρέφει το αρχείο “add-match.html”
- **Add Match()** : Λειτουργεί παρόμοια με την από πάνω μέθοδο. Χρησιμεύει στην ενημέρωση ενός ήδη υπάρχοντος αγώνα. Ζητάει από το front το συνθηματικό του αγώνα που θέλουμε να ενημερώσουμε και παίρνουμε όλο τον αγώνα με το get Match() που υλοποιείται στο Service. Επιστρέφει το αρχείο “add-match.html”

- **Show Matches()** : «Για το **μοντέλο** αγώνα φέρε μου τους αγώνες από το Service». Έτσι μπορεί να ερμηνευθεί η παραπάνω εντολή. Επιστρέφει το αρχείο “show-matches.html”
- **Delete Match()** : Ζητάω το τρέχον συνθηματικό του αγώνα(requestparam). Για το συγκεκριμένο id => delete match(). Ανακατεύθυνση στην φόρμα με τους αγώνες.
- **Show Odds()** : Κάθε αγώνας έχει κάποια/ες αποδόσεις. Εμφανίζουμε αυτές χάρη σ’ αυτήν τη μέθοδο. Τα **μοντέλα** μας είναι ο αγώνας και οι αποδόσεις του. Λόγω συσχέτισης χρειάζομαι πληροφορίες και των δύο, άρα για κάθε τρέχον αγώνα, θέλω και ζητάω το συνθηματικό του. «Τραβάω» τις αποδόσεις με βάση την μέθοδο get matchOdds(), η οποία παίρνει το id του τρέχοντος αγώνα και κάνει μέσω αυτού (λόγω συσχέτισης) .get ().getOdds (), δηλαδή παίρνει τον αγώνα => παίρνει τις αποδόσεις. Άρα τώρα έχω τις αποδόσεις του τρέχοντος αγώνα και χάρη στο model.addAtribute() τις εμφανίζω. Επιπλέον, κρατάω και το συνθηματικό αγώνα για μελλοντικά.
  Επιστρέφει το αρχείο “show-odds.html”
- **Add Odd()** : Ομοίως σαν το add match(), απλά κρατάω και το match id. Επιστρέφει το αρχείο “add-odd.html”.
- **Add Odd()** : Τη χρησιμοποιώ για να ανακατευθυνθώ εφόσον έχω περάσει μία απόδοση, στο show Odds(), κρατώντας το id και εμφανίζοντας το στο URL.
- **Create Match()** : Χωρίς το Model. Η μέθοδος αποθηκεύει έναν αγώνα χρησιμοποιώντας ως μοντέλο το ίδιο το αντικείμενο γι’ αυτό και η χρήση του @ ModelAttribute, σαν να μας λέει ότι το entity match θα είναι model με όνομα “match”.
- **Update Odd()** : Πρακτικά είναι σαν την add Odd(), αλλά εδώ αλλάζει ότι κρατάμε το συγκεκριμένο συνθηματικό της απόδοσης που θέλουμε να αλλάξουμε και του αγώνα εννοείται για να ξέρουμε σε ποιόν αγώνα απευθυνόμαστε. Γυρνάει στο αρχείο “add-odd.html”, έχουμε πάρει τα δεδομένα της απόδοσης και είναι ήδη συμπληρωμένη η φόρμα με τα τελευταία δεδομένα που είχε έτσι την ενημερώνουμε με αυτά που θέλουμε και στην τελική εκτελείται ακόμη μία add Odd().
- **Delete Odd()** : Ίδια με την Delete Match(), με την μόνη διαφορά ότι κρατάμε και το συνθηματικό του αγώνα για να ξέρουμε σε ποιόν αγώνα ανήκει αυτή που διαγράφουμε.

Περνάμε στο κομμάτι **Security**. Ένα τυπικό Security το οποίο δίνει πρόσβαση στα URLs. Τα annotations EnableWebSecurity Configuration, το πρώτο ενεργοποιεί το αυτοματοποιημένο security του spring για να αναγνωρίσει τις λειτουργίες του, ενώ το δεύτερο ορίζει ότι η κλάση είναι μια κλάση ρύθμισης.

```java
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

                http.authorizeHttpRequests(
                        (authz) -> authz
                        .requestMatchers("/**", "/add", "/save", "/showMatches", "/deleteMatch", "/showOdds", "/addOdd").permitAll()
                        .anyRequest().authenticated()
                        );
                return http.build();
    }

    @ControllerAdvice
    public class GlobalExceptionHandler {
        @ExceptionHandler(RuntimeException.class)
        public ResponseEntity<String> handleRuntimeException(RuntimeException ex) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(ex.getMessage());
        }
    }
}
```

Εικόνα 9. Controller

Security Filter Chain (): Η μέθοδος αυτή ορίζει σε ποιους είναι επιτρεπτά για πρόσβαση τα συγκεκριμένα URLs.

Global Exception Handler (): Διαχείριση εξαιρέσεων που προκύπτουν από κάποιον Controller. Το @ControllerAdvice δηλώνει ότι αυτή η κλάση είναι υπεύθυνη για global handling των εξαιρέσεων που σχετίζονται με controllers.

Το @ExceptionHandler(RuntimeException.class) ορίζει ότι αυτή η μέθοδος που το έχει θα καλείται όταν προκύψει μια εξαίρεση του τύπου RuntimeException.

Περνάω στο **View** κομμάτι του project.

VIEW.

Η αρχική μου σελίδα είναι μια τυπική σελίδα με ένα απλό κουμπί που δείχνει όλους τους αγώνες και επικοινωνεί με το /showMatches.

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Home Page</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5
    <style>
      .center-content {
        height: 100vh;
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
        text-align: center;
      }
    </style>
  </head>
<body>
  <div class="container center-content">
    <h1 class="mb-3">Trampas Eleftherios</h1>
    <p class="mb-4">trampasleuteris@gmail.com</p>
    <a th:href="@{/showMatches}" class="btn btn-primary btn-sm" th:text="'Matches'">Matches</a>
  </div>
</body>
</html>
```

Εικόνα 11. index.html (Για το homepage)

Όταν πατάω το κουμπί => **<a th:href="@{/showMatches}" class="btn btn-primary btn-sm" th:text="'Matches'">Matches**, δηλαδή εκτελείται το /showMatches από τον controller.

Εικόνα 12. showMatches

```
<title>Matches</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok
</head>
<body>
<div class="container my-4">

<h1 class="text-center mb-4">All The Matches</h1>

<table class="table table-striped">
<thead>
<tr>
<th scope="col">#</th>
<th scope="col">Description</th>
<th scope="col">Team A</th>
<th scope="col">Team B</th>
<th scope="col">Date</th>
<th scope="col">Time</th>
<th scope="col">Sport</th>
<th scope="col"></th>
</tr>
</thead>
<tbody>
<tr th:each="match, stat : ${matches}">
<th scope="row" th:text="${stat.index + 1}"></th>
<td th:text="${match.description}"></td>
<td th:text="${match.team_a}"></td>
<td th:text="${match.team_b}"></td>
<td th:text="${match.match_date}"></td>
<td th:text="${match.match_time}"></td>
<td th:text="${match.sport}"></td>
<td>
<a th:href="@{/updateMatch(matchId=${match.id})}" class="btn btn-info btn-sm">Update</a>
<a th:href="@{/showOdds(matchId=${match.id})}" class="btn btn-info btn-sm">Odds</a>
<a th:href="@{/deleteMatch(matchId=${match.id})}" class="btn btn-danger btn-sm" onclick="return confirm('Are you sure you want to delete
</td>
</tr>
</tbody>
```

Εικόνα 13. show-matches.html

Όταν **delete** => **<a th:href="@{/deleteMatch(matchId=${match.id})}" class="btn btn-danger btn-sm" onclick="return confirm('Are you sure you want to delete this match?');">Delete,** δηλαδή επικοινωνεί με το /deleteMatch και ορίζουμε το matchId που ζητάμε στον controller, τέλος θα έχουμε στο URL το αποτέλεσμα του id που θέλουμε.

Όταν **Odds** => **<a th:href="@{/showOdds(matchId=${match.id})}" class="btn btn-info btn-sm">Odds,** δηλαδή επικοινωνεί με το /showOdds και ορίζουμε το matchId που ζητάμε στον controller.

Όταν **Update** => **<a th:href="@{/updateMatch(matchId=${match.id})}" class="btn btn-info btn-sm">Update,** δηλαδή επικοινωνεί με το /updateMatch και ορίζουμε το matchId που ζητάμε στον controller.

```
<a href="/add" type="button" class="btn btn-primary">Add Match</a>
<a href="/" type="button" class="btn btn-primary">Home</a>
```

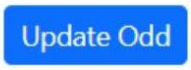
Όταν **Add Match**=> **Add Match**, απλό href που μας πηγαίνει στο /add του controller. Ομοίως και στο Home.

Στο add-odd.html χρησιμοποιήθηκε αυτή η γραμμή ώστε να παίρνουμε μαζί το id του τρέχοντος match.

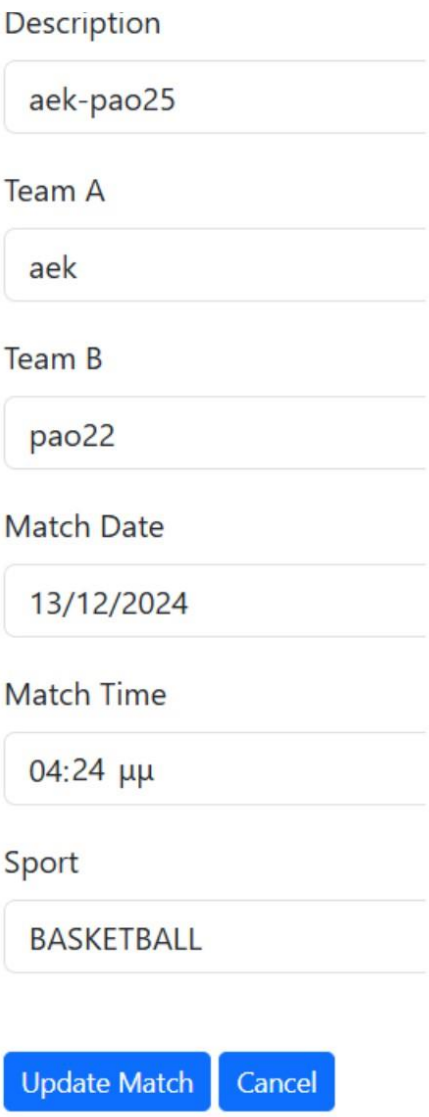
```
<br/>
<input type="hidden" th:field="*{id}" class="form-control" />
```

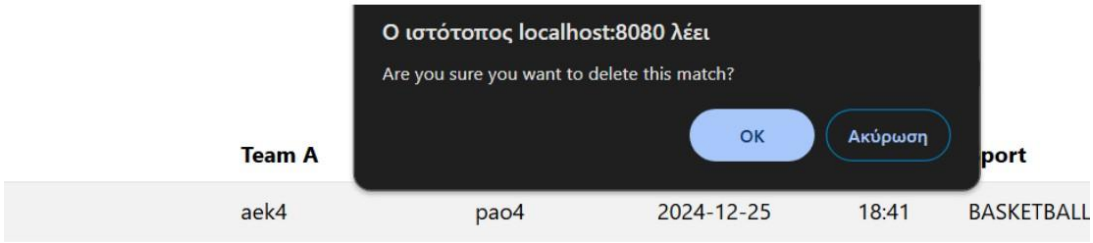
Η γενική ιδέα ακολουθείται και στις υπόλοιπες φόρμες .html. Τέλος, βοηθήθηκα με την βιβλιοθήκη bootstrap.

Παρακάτω έχουμε στιγμιότυπα από τη λειτουργεία της σελίδας.




Ο ιστότοπος localhost:8080 λέει
Are you sure you want to delete this odd?
OK
Ακύρωση
Specifier
1 x
2 1
3 2 3.4
Update Delete
Add Odd
Back to matches


Description
aek-pao25
Team A
aek
Team B
pao22
Match Date
13/12/2024
Match Time
04:24 μμ
Sport
BASKETBALL
Update Match
Cancel


Ο ιστότοπος localhost:8080 λέει
Are you sure you want to delete this match?
OK
Ακύρωση
Team A
aek4 pao4 2024-12-25 18:41 BASKETBALL

ΤΕΛΙΚΑ

Όσον αφορά το project υλοποιήθηκε από την Πέμπτη μέχρι την Δευτέρα ώστε να είναι λειτουργικό. Στο διάστημα Δευτέρας-Τετάρτης μεσολάβησαν κάποιες αλλαγές για την «καθαρότητα» του κώδικα και του view.

Ευχαριστώ εκ των προτέρων.