NHL - Fantasy League Application

Masters of Engineering - Electrical and Computer Engineering
ECE- 651 Foundation of Software Engineering - Group -1

Sneh Patel (sneh.patel@uwaterloo.ca)
Harshitkumar Patel (hv6patel@uwaterloo.ca)
Mohit Gupta (m52gupta@uwaterloo.ca)
Prateeti Deb Chaudhuri (pdebchau@uwaterloo.ca)
Priya Patel (pb4patel@uwaterloo.ca)

Overview

The NHL- Fantasy League is a form of fantasy sport that allows players to build a custom team and compete with other players, based on the statistics generated by professional hockey players or teams.

UNIVERSITY OF WATERLOO

Functional Properties

The Application implements the following functionalities:

1. Authentication: User Login and User logout.
2. User registration - Account creation for users.
3. User Functions - Update profile and view teams, upcoming matches and scores.
4. Team Functions - Create team, update team, separate teams in different leagues.
5. League Functions - Create and join leagues.
6. Scoring - Maintain the user scores for each league.
7. Player Background - Maintain the players data.

Non-Functional Properties

The application provides the following non-functional properties:

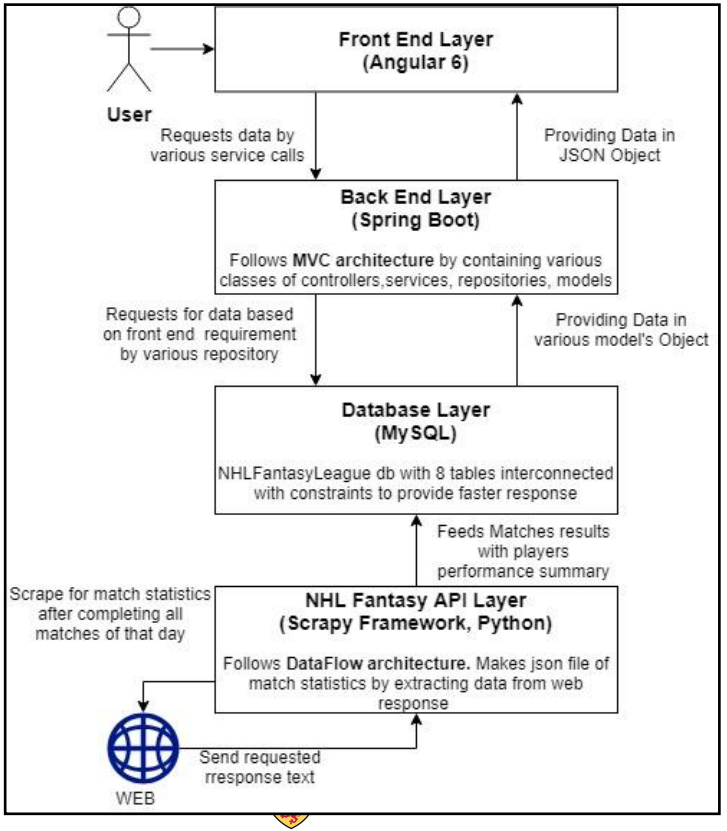
1. Security: the application is secure and no unauthorized user is allowed to access other users' data (users can only access their personal data, not anyone else's).
2. Data Persistence: data of user should be available and stored whenever the user requests for it.
3. Reusability: Back-end and front-end are loosely coupled. We can use both for another application.

Software Architectures

Layered Architecture

Consists of four layers:

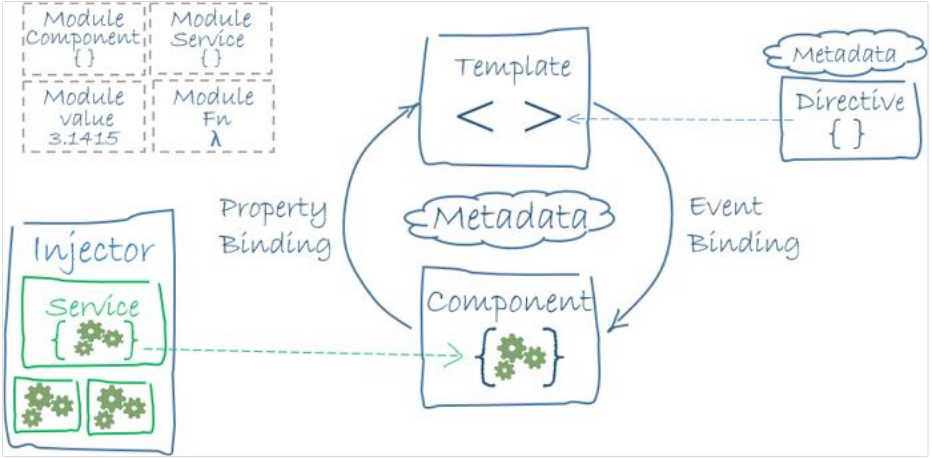
1. Front-end: User interactive layer implemented using Angular 6 with the Angular CLI tool.
2. Back-end: Implements business logic using Spring Boot framework.
3. Data layer: It consists of database tables and their mappings.
4. API layer: Follows dataflow architecture to extract data from the web.



Event-based Architecture

Architecture involves two parts:

1. Listening/binding: Binding the function to the event.
2. Announcing/triggering : Makes the event or triggering or firing an event.

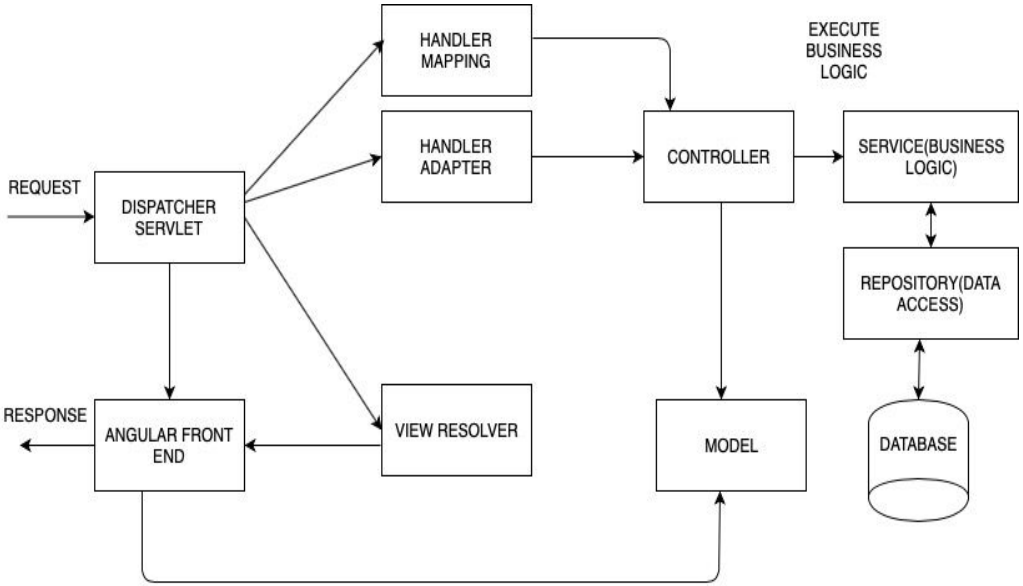


UNIVERSITY OF WATERLOO

Spring MVC Architecture

MVC mainly consists of:

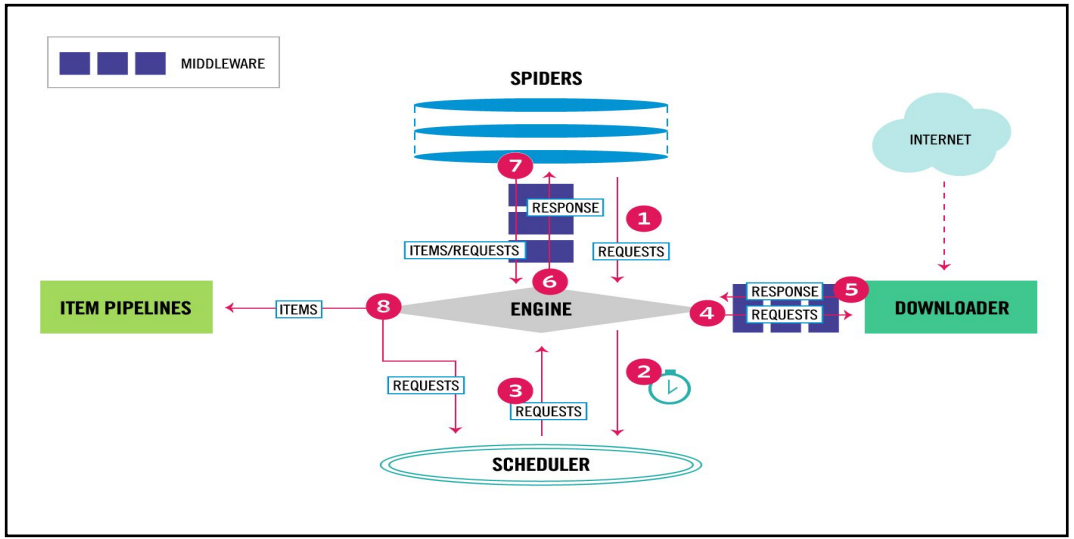
1. **Model:** encapsulates the application data (POJO).
2. **View:** renders the model data.
3. **Controller:** Receives the user request, builds a model object and pass it to the view.



UNIVERSITY OF WATERLOO

Dataflow Architecture

NHL fantasy API feeds data to the database layer using Scrapy framework in Python.



UNIVERSITY OF WATERLOO

Design Patterns

Design Patterns

1. Dependency Injection

```
@RestController
public class UserController
{
        @Autowired
        UserService userService;
```

Dependency injected service into controller

```
@RequestMapping(value = "/api/register", method = RequestMethod.POST,
consumes = "application/json", produces = "application/json")
        public @ResponseBody User addUser(@RequestBody User user,
HttpServletResponse response, HttpServletRequest request) {
```

```
export class UserComponent implements OnInit {
  user: User;
  id: number;
  inactive: boolean = true;
  editButton: boolean = false;
  submitButton: boolean = true;
  joinLeagueForm: boolean = false;
  joinLeagueButton: boolean = true;

  constructor(private userService: UserService,
      private route: ActivatedRoute,
          private router: Router) { }
```

Service injected into component

```
@Injectable({
  providedIn: 'root'
})
export class UserService {

  constructor(private http: Http) { }
```

UNIVERSITY OF WATERLOO

Design Patterns (contd.)

2. Singleton

```
class Singleton:
    __instance = None
    @staticmethod
    def getInstance():
        if Singleton.__instance == None:
            Singleton()
        return Singleton.__instance
    def __init__(self):
        if Singleton.__instance != None:
            raise Exception()
        else:
            Singleton.__instance = self
    def addPlayersToDb(self,fileName,dbPassword)
    def
addScheduleToDb(self,fileName,dbPassword)
```

getInstance()
creates the object
(if it does not exist)
and returns it

```
s = Singleton.getInstance()
        s.addPlayersToDb(playersF
ile,
                    args.dbPassword)
        s.addScheduleToDb(schedul
eFile,
                    args.dbPassword)
```

UNIVERSITY OF WATERLOO

Design Patterns (contd.)

3. Observer

```
this.userService.getUser(usr).subscribe(data => {
  this.user = data;
});
```

Observer subscribes to the observable returned by HTTP client

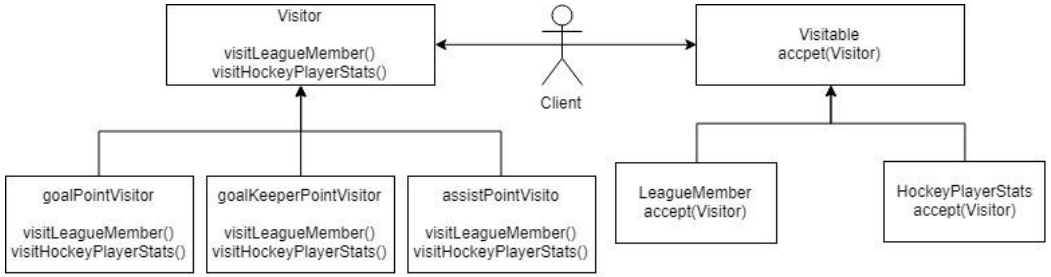
```
getUser(usr){

  let tempObj = {
    "userid": usr.userid,
  }

  return this.http.post('/api/user', tempObj).
    pipe(map((response: Response) => response.json()));
}
```

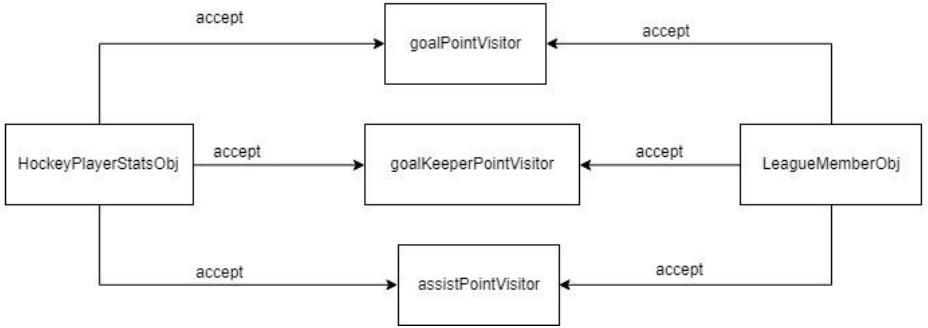
UNIVERSITY OF WATERLOO

Design Patterns (contd.)

4. Visitor

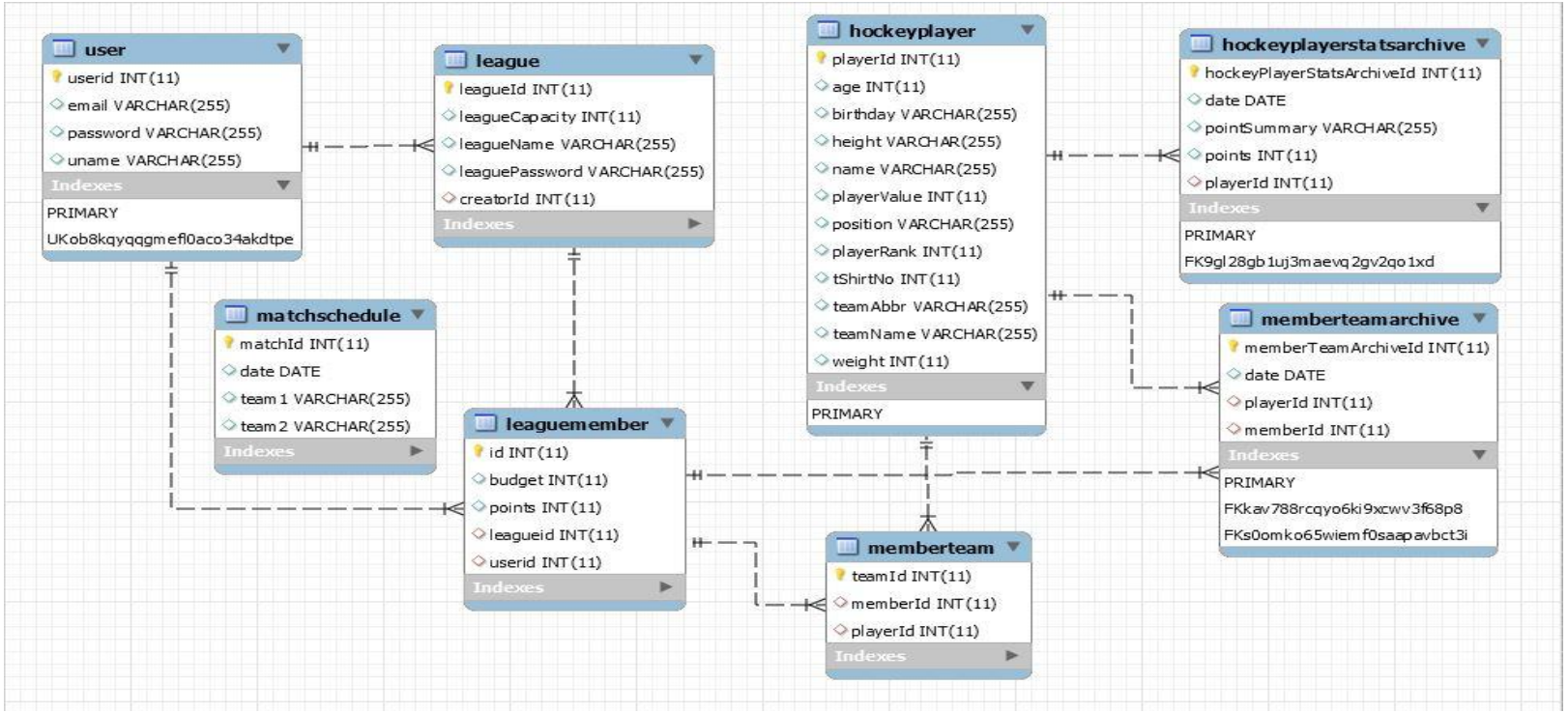
The visitors (goal, assist and goalkeeper's saves and goal against) visits each league member and hockey player statistics and adds points to the objects hockeyPlayerStatsObj and LeagueMemeberObj which accepts the visitors





UNIVERSITY OF WATERLOO

Database Design



UNIVERSITY OF WATERLOO

Let’s move on to the demo!