Backend

package.json

```
{
  "name": "tdk-app-backend",
  "version": "1.0.0",
  "description": "",
  "main": "main.js",
  "dependencies": {
    "D": "^1.0.0",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "http": "0.0.0",
    "https": "^1.0.0",
    "mongodb": "^3.5.5",
    "underscore": "^1.10.2"
  },
  "devDependencies": {
    "@types/express": "^4.17.6",
    "@types/express-session": "^1.17.0",
    "@types/jest": "^24.9.1",
    "@types/mongodb": "^3.5.5",
    "@types/node": "^12.12.31",
    "@types/passport": "^1.0.3",
    "@types/passport-http": "^0.3.8",
    "@types/supertest": "^2.0.8",
    "jest": "^24.9.0",
    "supertest": "^4.0.2",
    "ts-jest": "^24.3.0",
    "typescript": "^3.6.3"
  },
  "scripts": {
    "test": "jest",
    "build": "tsc --target ES5 src/main.ts --outDir dist",
    "test-coverage": "npm test -- --coverage"
  },
  "jest": {
    "transform": {
      ".(ts|tsx)": "<rootDir>/node_modules/ts-jest/preprocessor.js"
    },
    "testRegex": "(/__tests__/.*|\\.(test|spec))\\.(ts|tsx|js)$",
    "moduleFileExtensions": [
      "ts",
      "tsx",
      "js"
    ]
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

dependecies: az egyes dependecy-k, az npm install-al ezek kerülnek majd fel, illetve fejlesztés közben ha npm install <dependency_name> akkor a package.json-ban is megjelenik a verzióval együtt. express, mongodb kell, az express a kérésekhez (post, get, delete stb) a mongodb az adatbázishoz.

devDependency: nekem a typescript miatt az egyes @types/<name> kellett azért, hogy felhozza az egyes függvényeket a VSCode.

scripts: ide lehet script-eket létrehozni, teszt, build script-eket, nekem a build kellett, itt a ts-ből a js fájlokat máshova teszi.

src/app.ts

```
app.post('/applications', async function (req, res) {
    const application: Application = new Application(req.body);
    let errors: Error[] = [];



    errors.push.apply(errors, Validator.applicationsPostError(application));
    console.log(errors);
    if (errors.length > 0) {
        console.log(errors);
        return res.status(500).send(errors);
    }
    try {
        await insertApplication(application);
    } catch (e) {
        console.log(e);
        return res.status(500).send(e);
    }
    return res.status(200).send();

});
```

egy post kérést valósít meg, a jelentkezéseket töltjük fel a db-be. Egy új Application objektumot létehozunk (a body-ban várjuk az application-t). Az Error egy saját hibakezelő, ellenőrzést végzek (ne legyenek üres értékek). Ha hiba van (errors.length>0) akkor 500-as értékkel térek vissza, ha nincs hiba, akkor az insertApplication()-t hívom meg, ami beszúrja a db-be az adatokat, majd ha a beszúrás során nincs hiba, akkor 200-al térek vissza.

```
app.get('/applications', async function (req, res) {
    let applications: Application[] = [];
    try {
        //await createDB();
        //await createApplication();
        applications = await listApplication();
    } catch (e) {
        console.log(e);
        return res.status(500).send(e);
    }
    return res.status(200).send(applications);
});
```

Az a get kérést valósítja meg, az összes Application-t kéri le. Meghívom a listApplication() függvényt, ami a db-ből listázza az application-t. Ha hiba van akkor 500-al térek vissza, ha nincs hiba akkor 200-al és az Application[]-el.

```
app.put('/applications', async function (req, res) {
    const application: Application = new Application(req.body);
    let errors: Error[] = [];
    errors.push.apply(errors, Validator.applicationsPostError(application));
    if (errors.length > 0) {
        console.log(errors);
        return res.status(500).send(errors);
    }
    try {
```

```
        await updateApplication(application);
    } catch (e) {
        return res.status(500).send(e);
    }
    return res.status(200).send();
});
```

A put kérést valósítása, ahol request body-ba kapjuk meg az Application-t, leellenőrizzük, hogy helyes-e majd ha helyes akkor update-eljük az updateApplication()-t meghívjuk. Ha sikerült az update, akkor 200-al térünk vissza.

```
app.delete('/applications/:applicationId', async function (req, res) {
    const applicationId = req.params.applicationId;
    try {
        await deleteApplication(applicationId);
    } catch (e) {
        return res.status(500).send(e);
    }
    return res.status(200).send();

});
```

A delete kérést valósítja meg. Az applicationId-t paraméterként kapja meg, és azt id alapján töröl (meghívja a deleteApplication-t). Ha rendben ment a törlés, akkor 200-al tér vissza.

Validator.ts

Itt értékellenőrzés van, az Error egy saját osztály, locationProperty és message-el, azt jelzi hogy hol és mi a hiba.

Model/…ts

A model osztályok

ApplicationService.ts

Egy réteg a MongoService felett, így kicserélhető a db úgy, hogy csak itt kell átírni a kódot, és nyilván a MongoService-t le kell cserélni pl MySQLService-re stb..

Mivel a MongoService általánosan van megírva, így az egyes query-ket az ApplicationService-ben kell elkészíteni, pl:

```
export async function createApplication() {
    await mongoService.createCollection("Application");
}
export async function insertApplication(application: Application) {
    await mongoService.insertOneCollection("Application", application);
}

export async function listApplication(): Promise<Application[]> {
    return await mongoService.listCollection("Application", {}, {});
}
export async function updateApplication(application: Application) {
```

```
    await mongoService.updateOneCollection("Application", { _id: new ObjectId(application._id) }, { $set: { status: a
pplication.status } });
}
export async function deleteApplication(applicationId: string) {
    await mongoService.deleteOneCollection("Application", { _id: new ObjectId(applicationId) });
}
```

1. kollekció (Application) létrehozás

2. Egy új kollekció (Application) felvitele

3. Kollekció listázása (Application)

4. Kolleció update-elése (Application), id alapján (`{ _id: new ObjectId(application._id) }, )`, itt csak a státuszt állítjuk (`{ $set: { status: application.status } )`

5. Kollekció törlése (Application), id alapján (`{ _id: new ObjectId(application._id) }, ),`

MongoService.ts

DB-vel kapcsolatos műveletek, pl listázás, törlés, update stb.

```
listCollection(collectionName: string, query1: any, query2: any): Promise<any> {
        return new Promise((resolve, reject) => {
            MongoClient.connect(this.url, {
                useUnifiedTopology: true,
                useNewUrlParser: true,
            })
                .then((db) => {
                    var dbo = db.db(this.databaseName);

                    return dbo.collection(collectionName).find(query1, query2).toArray().then((c
ollection) => {
                        db.close();
                        resolve(collection);
                    }).catch(err => {
                        console.log(`DB Connection Error: ${err.message}`);
                        db.close();
                        reject(err.message);
                    }).finally(() => {
                        console.log('Close DB');
                        //db.close();
                    })
                });
        });
    }
```

a find()-al lehet listázni, a collectionName, query1 és query2-t a DBService adja meg, itt nekünk most Applicaiton lesz és {}, tehát mindent listázunk. A DB-hez kapcsolódunk mad a parancsot kiadjuk a db-nek, hibakezelés és visszatérünk a collection-el amit listázott (`resolve(collection);`)

A többi művelet is hasonló, csak :

```
dbo.collection(collectionName).updateOne(query, newValues)
```

```
dbo.collection(collectionName).deleteOne(query)
```

```
dbo.collection(collectionName).drop()
```

kb ennyiben különbözik a fenti kódtól.

Frontend

package.json

Mint backend leírásban…

app.module.ts

Itt vannak a komponensek, modul-ok, service-ek deklarálva:

Nyilván itt import-álni is kell őket, + az alábbi helyeken feltüntetni:

komponenseket

```
declarations: [
    AppComponent,
    UserAppylingTdkComponent,
    AdministratorMenuConferenceListParticipantsComponent,
  ],
```

modul-okat:

```
imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
```

pl itt van nekem egy jópár material-os modulom.

És a service-eim is itt vannak:

```
providers: [
    { provide: MAT_FORM_FIELD_DEFAULT_OPTIONS, useValue: { appearance: 'fill' } },
    MatDatepickerModule, ApplicationService],
  bootstrap: [AppComponent],
})
```

index.html

Ide tettem a Material-hoz szükséges import-ot

```
  <link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500&display=swap" rel="stylesheet">
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
```

app.component.html

```
<mat-drawer-container class="page-container">
  <mat-drawer mode="side" opened class="page-nav">
    <mat-nav-list>

      <a mat-list-item routerLink="/conference/list/participants"><span class="material-icons">
          list
        </span>Jelentkezettek listázása</a>

      <a mat-list-item routerLink="/user/applyingTDK"><span class="material-icons">
          add_circle
        </span>Jelentkezés</a>
```

```
    </mat-nav-list>
  </mat-drawer>
  <mat-drawer-content class="page-content">
    <router-outlet></router-outlet>
  </mat-drawer-content>
</mat-drawer-container>
```

Ez a menü-t csinálja meg material-osan, 2 elem közül lehet választani a konferencia résztvevő listázás (`="/conference/list/participants"`), és a jelentkezés (`="/user/applyingTDK"`)

2 komponensem van, ezt fogjuk most megnézni:

user-applying-tdk komponens

html fájl:

```
<mat-divider></mat-divider>
```

ez csak gyakorlatilag egy vonal Material-osan

```
<mat-form-field>
    <input name="application.tdkTitle" matInput [(ngModel)]="application.tdkTitle" placeholder="A pályamunka címe"
      type="text" required minlength="1" #applicationTdkTitleModel="ngModel" />

    <mat-error
      *ngIf="applicationTdkTitleModel.invalid && (applicationTdkTitleModel.dirty|| applicationTdkTitleModel.touched)"
>
      A pályamunka címét
      kötelező megadni</mat-error>

  </mat-form-field>
```

Itt egy input-ban a pályamunka címét adja meg a felhasználó. nyilván <mat-…> a material-os dolgok. az application tdkTitle-t kötöttem rá (a ts fájl objektuma), a `#applicationTdkTitleModel="ngModel"` itt azért kell, hogy ellenőrizzem a z input értékét, állapotát és hibát írjak.

Egy lenyíló lista létrehozása:

```
<mat-form-field>
    <mat-label>Kar</mat-label>
    <mat-select name="application.facultyName" [(ngModel)]="application.facultyName">
      <mat-option *ngFor="let facultyOption of faculties" [value]="facultyOption.value">
        {{facultyOption.value}}
      </mat-option>
    </mat-select>
  </mat-form-field>
```

Itt az application FacultyName-je fogja tartalmazni az értéket. ÉS a faculties-en megyünk végig a lenyíló lista elemei ezek leszek.

Táblázatot is tartalmaz a fájl, ezt fogjuk most megnézi:

```
<table #supervisorsTable mat-table [dataSource]="createdSupervisorsSource" class="mat-elevation-z8">
  <ng-container matColumnDef="name">
    <th mat-header-cell *matHeaderCellDef>Név</th>
    <td mat-cell *matCellDef="let supervisor">{{supervisor.name}} </td>
  </ng-container>
```

a dataSource a ts fájlban lévő createdSupervisorsSource:

```
createdSupervisorsSource: MatTableDataSource<Supervisor> = new MatTableDataSource([]);
```

ezek is kellenek a ts fájlban hozzá:

```
 @ViewChild('supervisorsTable', { static: false }) supervisorsTable: MatTable<any>;
  displayedColumnsSupervisors: string[] = ['name', 'position', 'faculty', 'institute', 'deleteSupervisor'];
  dataSourceSupervisors = new MatTableDataSource(this.createdSupervisors);
```

a táblázatot @ViewChild-al tudjuk elérni a ts fájlban, a displayedColumsSupervisors az oszlopok azonosítói, a dataSourceSuoervisors pedig a supervisor-ok, azaz ezek lesznek a táblázat elemei.

A legvégén fel is kell vinni a jelentkezést, a gomb:

```
button mat-raised-button color="primary" (click)="applyingTDK()"
  [disabled]="applicationTdkTitleModel.invalid||applicationFacultySectionNameModel.invalid"><span
    class="material-icons">
    add
  </span>Jelentkezés</button>
```

Itt ha minden rendben van akkor nem disabled, a lekezelő az applyingTDK() ami a ts kódban van ugye.

ts fájl

létrehozunk minden elemnek amit fel kell vinni a form-ban egy-egy objektumot:

```
application: Application = new Application();

  supervisor: Supervisor = new Supervisor();
```

Ez csak annyi, hogy a lenyíló listának és a radio button-nak van egy-egy konstans definiálva, és itt a property-nek értékül adom

```
 studies = studies;
  faculties = faculties;
  taxation = taxations;
  typeOfTheSpecialization = studies;
```

Ezek a konstansok így néznek ki (Study.ts)

```
interface Study {
    value: string;
    viewValue: string;
}
export const studies: Study[] = [
    { value: 'BSc', viewValue: 'BSc' },
    { value: 'MSc', viewValue: 'MSc' },
    { value: 'Osztatlan', viewValue: 'Osztatlan' },
    { value: 'FOSZ', viewValue: 'FOSZ' },
    { value: 'Bprof', viewValue: 'Bprof' },
];
```

A következő sorok a táblázathoz kellenek (fentebb írtam róla a html-nél):

```
  @ViewChild('applyingTDKForm', { static: false, read: ElementRef }) applyingTDKForm: ElementRef;
  @ViewChild('supervisorForm', { static: false, read: ElementRef }) supervisorForm: ElementRef;
  @ViewChild('authorForm', { static: false, read: ElementRef }) authorForm: ElementRef;
  @ViewChild('supervisorsTable', { static: false }) supervisorsTable: MatTable<any>;
  displayedColumnsSupervisors: string[] = ['name', 'position', 'faculty', 'institute', 'deleteSupervisor'];
  dataSourceSupervisors = new MatTableDataSource(this.createdSupervisors);
  @ViewChild(MatPaginator, { static: true }) paginator: MatPaginator;
  displayedColumnsAuthors: string[] = ['name', 'facultyName', 'specialization', 'year', 'neptunCode', 'idNumber', 'ty
peOfTheSpecialization', 'taxStatus', 'isEmployee', 'taxIdentificationNumber', 'birthPlace', 'birthDate', 'nameOfTheMo
ther', 'zipCode', 'town', 'streetAndNumber', 'telephoneNumber', 'email', 'bankAccountNumber', 'deleteAuthor'];
  dataSourceAuthors = new MatTableDataSource(this.createdAuthors);
  @ViewChild(MatPaginator, { static: true }) paginatorAuthors: MatPaginator;
  @ViewChild('TableSupervisorPaginator', { static: true }) tableSupervisorPaginator: MatPaginator;
  @ViewChild('TableAuthorPaginator', { static: true }) tableAuthorPaginator: MatPaginator;
```

```
initApplicationValues()
```

csak az egyes értékeket inicializálom, a rádió gombnál az egyik checked-legyen, lenyíló listánál egy legyen kiválasztva mikor a felhasználó meglátja a form-ot stb.

```
applyingTDK()
```

amikor már minden felvittünk és jelentkezünk. leellneőrzi a jelentkezést, majd az applicationService-t hívja meg. A service fogja kiadni a kérést. Ez így néz ki:

```
this.applicationService.addApplication(this.application).subscribe(
      () => {
        this.initApplicationValues();
        this.applyingTDKForm.nativeElement.reset();
        this.createdSupervisorsSource.data = [...this.createdSupervisors];
        this.createdAuthorsSource.data = [...this.createdAuthors];
      }, (error) => {
        alert('Hiba a jelentkezés során');
      }
    );
```

feliratkozik rá, és ha a kérés megvan utána a form-okat reseteli,újra init-be hozza a form-ot, a createdSupervisorDataSorce stb (ez a táblázat elemei, amiket felvitt a felhasználó) szintén beállítja (üres lesz itt az értéke a createdSuperVisors üres tömb lesz az initApplication() miatt).

Új Supervisor felvitele a táblázatba:

```
addNewSupervisor() {
    this.createdSupervisors.push(this.supervisor);
    this.supervisor = new Supervisor();
    this.supervisorForm.nativeElement.reset();
    //this.dataSourceSupervisors.data = this.createdSupervisors;
    console.log(this.createdSupervisors);
    this.createdSupervisorsSource.data = [...this.createdSupervisors];
    //this.supervisorsTable.renderRows();
  }
```

Itt a createdSupervisor[] eleme lesz a form Supervisor-ja és a form-ot reset-eljük, és a dataSource-t álltjuk a táblázat miatt.

Supervisor törlése (táblázatból és a tömbből)

```
removeSupervisor(supervisor: Supervisor) {
    this.createdSupervisors = this.createdSupervisors.filter(actualSupervisor => actualSupervisor != supervisor);
    this.createdSupervisorsSource.data = [...this.createdSupervisors];
  }
```

Itt a createdSupervisor-t állítjuk (úgyis ezt fogjuk majd az Application-nek a legvégén átadni, majd a backend-nek elküldeni az Application-t) és a táblázatot frissítjük.

Services/ApplicationService.ts

Gyakorlatilag itt fogjuk a backend-hez a kéréseket kiadni:

```
@Injectable()
export class ApplicationService {
    applications: Application[] = [];
    baseURL: string = 'http://localhost:3000';
    constructor(private http: HttpClient) {

    }

    addApplication(application: Application): Observable<Application> {
        return this.http.post<Application>(this.baseURL + '/applications', application);
    }
    updateApplication(application: Application): Observable<Application> {
        return this.http.put<Application>(this.baseURL + '/applications', application);
    }
    deleteApplication(application: Application): Observable<Application> {
        return this.http.delete<Application>(this.baseURL + '/applications/' + application._id);
    }
    listApplication(): Observable<Application[]> {
        return this.http.get<Application[]>(this.baseURL + '/applications');
    }

}
```

Injectable jelzi, hogy service,a HttpClient a http kéréshez kell. Itt adjuk át nyilván az egyes kérésekhez a paramétereket is.

Nézzük meg most a másik komponenst:

administrator-menu-conference-list-participants:

htlm fájl:

```
<mat-form-field>
  <mat-label><span class="material-icons">
      search
    </span>Keresés</mat-label>
  <input matInput (keyup)="applyFilter($event)" placeholder="Keresés">
</mat-form-field>
<br>
```

ez a keresést valósítja meg. lehet keresni a táblázatban

Itt a státuszt lehet állítani, elfogadott, elutasított, és törölni is lehet jelentkezést

```
<mat-accordion>
```

```
        <mat-expansion-panel>
          <mat-expansion-panel-header>
            <mat-panel-title>
              Jelentkezés művelet
            </mat-panel-title>
            <mat-panel-description>

            </mat-panel-description>
          </mat-expansion-panel-header>
          <button mat-raised-button color="basic" (click)="setStatusAccepted(application)"
            class="expansion-button">Jóváhagyás</button> <br><br>
          <button mat-raised-button color="basic" (click)="setStatusRejected(application)"
            class="expansion-button">Elutasítás</button>
          <button mat-raised-button color="basic" (click)="deleteApplication(application)"
            class="expansion-button">Törlés</button>
        </mat-expansion-panel>
      </mat-accordion>
```

Ez egy ilyen expansion-panel-ben van , erre gondoltam (https://stackblitz.com/angular/qxrbxnbpava?file=src%2Fapp%2Fexpansion-overview-example.ts), így egy kicsit jobban néz ki (talán ☺)

Nézzük még meg a táblázatból a témavezetőket: egy jelentkezéshez több témavezető (és szerző) is társulhat.

```
<ng-container matColumnDef="supervisors">
    <th mat-header-cell *matHeaderCellDef> Témavezető(k) </th>
    <td mat-cell *matCellDef="let application">
      <span *ngFor="let supervisor of application.supervisors">
        <mat-accordion>
          <mat-expansion-panel>
            <mat-expansion-panel-header>
              <mat-panel-title>
                Témavezető adatai
              </mat-panel-title>
              <mat-panel-description>
                {{supervisor.name}}
              </mat-panel-description>
            </mat-expansion-panel-header>
            <p>Neve: {{supervisor.name}}</p>
            <p> Beosztása: {{supervisor.position}}</p>
            <p> Munkahelye: {{supervisor.faculty}} {{supervisor.institute}}</p>
          </mat-expansion-panel>
        </mat-accordion>
      </span>
    </td>
  </ng-container>
```

Ezért ngFor-al járjuk meg a témavezetők tömbjét. Itt is az expansion-panel-t használtam, hogy ne írjon ki minden adatot.

TS fájl:

teljesen hasonló a jelentkezős fájlhoz, itt is van a táblázathoz köthető material-os kód, van service hívása stb.

Táblázathoz ezek kellenek:

```
displayedColumns: string[] = ['operations', 'status', 'tdkTitle', 'facultySectionName', 'facultyName', 'equipments',
'supervisors', 'authors'];
  dataSource = new MatTableDataSource(this.applications);
  @ViewChild(MatPaginator, { static: true }) paginator: MatPaginator;
```

És a filter (ha szeretnénk a táblázathoz):

```
 applyFilter(event: Event) {
    const filterValue = (event.target as HTMLInputElement).value;
    this.dataSource.filter = filterValue.trim().toLowerCase();
    console.log('itt');
  }
```

A getApplication()-el tudjuk lekérni a service segítségével a jelentkezéseket:

```
getApplications() {
    this.applicationService.listApplication().subscribe(applications => {
      this.applications = applications;
      this.dataSource.paginator = this.paginator;
      this.dataSource = new MatTableDataSource(this.applications);
    }, () => {
      alert("Hiba történt a konferencia listázás során");
    }
    );
  }
```

A táblázatot is nyilván állítjuk ilyenkor.

Ha update-eljük az application-t:

```
setStatusAccepted(application: Application) {
    application.status = ApplicationStatus.ACCEPTED;

    this.applicationService.updateApplication(application).subscribe(() => {
      alert("A jelentkezés módosítása sikeres");
      this.getApplications();

    }, () => {
      alert("Hiba történt a jelentkezés módosítása során");
      this.getApplications();
    }
    );

  }
```

a service updateApplciation()-t hívjuk meg, majd lekérjük az adatokat (az update után).

A delete során töröljük az alkalmazást:

```
deleteApplication(application: Application) {
    this.applicationService.deleteApplication(application).subscribe(() => {
      alert("A jelentkezés törlése sikeres");
      this.getApplications();

    }, () => {
      alert("Hiba történt a jelentkezés törlése során");
      this.getApplications();
    }
    );

  }
```

Itt töröljük az application-t, majd nyilván újra lekérhetjük hogy mik vannak a db-ben.

Egy dolgot elfelejtettem említeni a service-ekről, be kell injektálni őket ahol használni szeretnénk, pl ebben a ts fájlban:

```
constructor(private applicationService: ApplicationService) { }
```