TARA

Eleonora Bogdanova, Kat Garnier, Justin Griswold, Tori Jameson, Ave Kaye (mentor)

Description

General to severe, anxiety is a common condition with the ever increasing pressures of modern American life.

Tara is a unique app designed to help users track their triggers, needs, and occurrences with an eye toward helping the loved ones of those struggling with anxiety better assist them.

Features

- Customized User Registration and Login - shown on homepage
- User Needs and Triggers Survey that links to Search Filter
- Display of Survey Results on User Page
- Interface for Logging of Anxiety Occurrences
- Display page for Anxiety Occurrences
- Search and sorting functionality for occurrence log
- Links to crisis numbers and ability to call services within app

Planning - User Stories

As a user with anxiety, I want to track my anxiety occurrences so that I and people who love me can better understand patterns within those occurrences.

As a loved one of someone with anxiety, I want to be able to see the triggers of my anxious loved one and what helps (and doesn't help) them when they are anxious so that I may better care for them.

As someone who is in community with people with anxiety, I want to be able to reach safer appropriate and affirming professional help so that if help beyond me is needed I can act.

Planning - Database

- Users - ID, username, hashed password, and email
- Roles - Determine if the user is a standard user or supporter
- Tags - Stressors, helpers, and “don’ts”
    - Sub tables connect specific categories of tags to user IDs
- Log - User id, date, location, description, stressors, and helpers

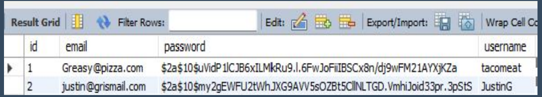
Technology Stack

- Angular 13
- Spring Boot
- Java 11
- Hibernate
- MySQL
- Bootstrap

What We Learned

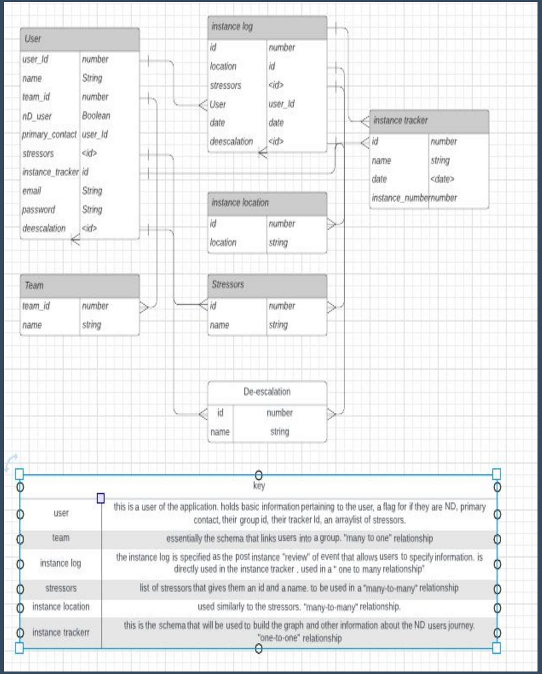
- Spring Security
- JSON Web Tokens (JWTs)
- Using an API to connect front and back ends
- Implementing components from Angular Material
- Filtering and sorting algorithmic methodology
- Wireframing, Kanban / Agile Methods
- Team Adaptation around Health and Family Needs

Password hashing



	id	email	password	username
▶	1	Greasy@pizza.com	$2a$10$uVidP1lCJB6xILMkRu9.l.6FwJoFiiIBSCx8n/dj9wFM21AYXjKZa	tacomeat
	2	justin@grismail.com	$2a$10$my2gEWFU2tWhJXG9AVV5sOZBt5CllNLTGD.VmhiJoid33pr.3pStS	JustinG

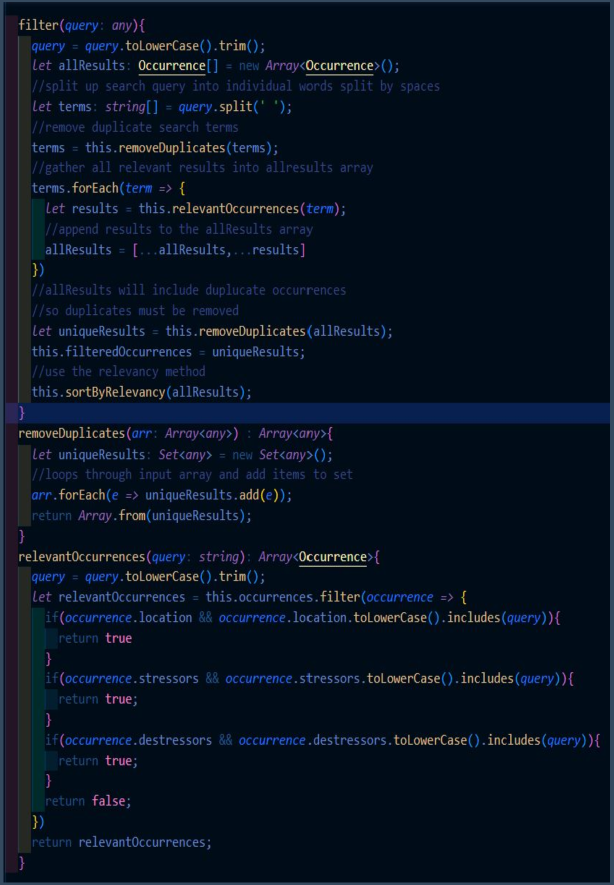
Data Schema



key

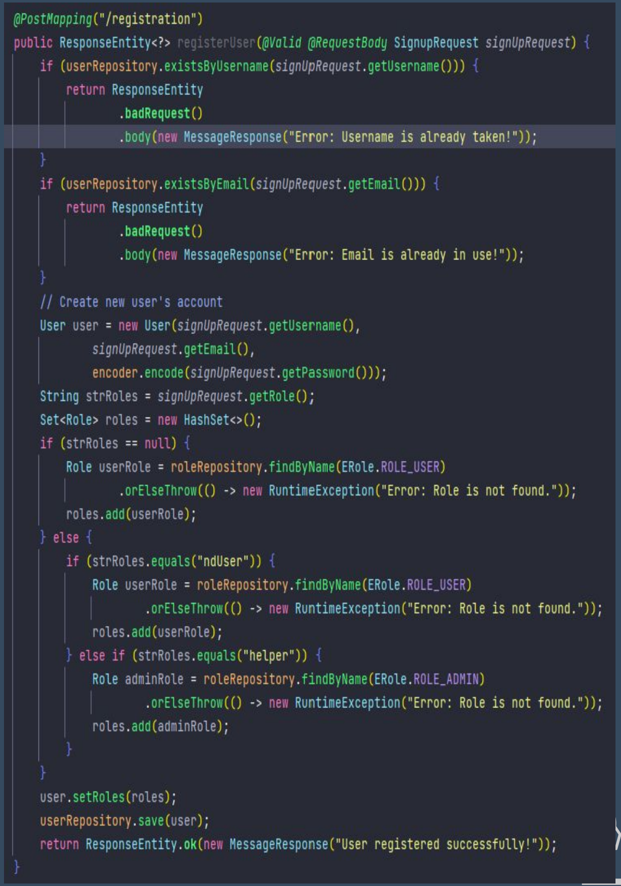
user	this is a user of the application. holds basic information pertaining to the user, a flag for if they are ND, primary contact, their group id, their tracker Id, an arraylist of stressors.
team	essentially the schema that links users into a group. "many to one" relationship
instance log	the instance log is specified as the post instance "review" of event that allows users to specify information. is directly used in the instance tracker . used in a " one to many relationship"
stressors	list of stressors that gives them an id and a name. to be used in a "many-to-many" relationship
instance location	used similarly to the stressors. "many-to-many" relationship.
instance trackerr	this is the schema that will be used to build the graph and other information about the ND users journey. "one-to-one" relationship

Log filter



```
filter(query: any){
  query = query.toLowerCase().trim();
  let allResults: Occurrence[] = new Array<Occurrence>();
  //split up search query into individual words split by spaces
  let terms: string[] = query.split(' ');
  //remove duplicate search terms
  terms = this.removeDuplicates(terms);
  //gather all relevant results into allresults array
  terms.forEach(term => {
    let results = this.relevantOccurrences(term);
    //append results to the allResults array
    allResults = [...allResults,...results]
  })
  //allResults will include duplucate occurrences
  //so duplicates must be removed
  let uniqueResults = this.removeDuplicates(allResults);
  this.filteredOccurrences = uniqueResults;
  //use the relevancy method
  this.sortByRelevancy(allResults);
}
removeDuplicates(arr: Array<any>) : Array<any>{
  let uniqueResults: Set<any> = new Set<any>();
  //loops through input array and add items to set
  arr.forEach(e => uniqueResults.add(e));
  return Array.from(uniqueResults);
}
relevantOccurrences(query: string): Array<Occurrence>{
  query = query.toLowerCase().trim();
  let relevantOccurrences = this.occurrences.filter(occurrence => {
    if(occurrence.location && occurrence.location.toLowerCase().includes(query)){
      return true
    }
    if(occurrence.stressors && occurrence.stressors.toLowerCase().includes(query)){
      return true;
    }
    if(occurrence.destressors && occurrence.destressors.toLowerCase().includes(query)){
      return true;
    }
    return false;
  })
  return relevantOccurrences;
}
```

Post mapping registration



```
@PostMapping("/registration")
public ResponseEntity<?> registerUser(@Valid @RequestBody SignupRequest signUpRequest) {
    if (userRepository.existsByUsername(signUpRequest.getUsername())) {
        return ResponseEntity
                .badRequest()
                .body(new MessageResponse("Error: Username is already taken!"));
    }
    if (userRepository.existsByEmail(signUpRequest.getEmail())) {
        return ResponseEntity
                .badRequest()
                .body(new MessageResponse("Error: Email is already in use!"));
    }
    // Create new user's account
    User user = new User(signUpRequest.getUsername(),
            signUpRequest.getEmail(),
            encoder.encode(signUpRequest.getPassword()));
    String strRoles = signUpRequest.getRole();
    Set<Role> roles = new HashSet<>();
    if (strRoles == null) {
        Role userRole = roleRepository.findByName(ERole.ROLE_USER)
                .orElseThrow(() -> new RuntimeException("Error: Role is not found."));
        roles.add(userRole);
    } else {
        if (strRoles.equals("ndUser")) {
            Role userRole = roleRepository.findByName(ERole.ROLE_USER)
                    .orElseThrow(() -> new RuntimeException("Error: Role is not found."));
            roles.add(userRole);
        } else if (strRoles.equals("helper")) {
            Role adminRole = roleRepository.findByName(ERole.ROLE_ADMIN)
                    .orElseThrow(() -> new RuntimeException("Error: Role is not found."));
            roles.add(adminRole);
        }
    }
    user.setRoles(roles);
    userRepository.save(user);
    return ResponseEntity.ok(new MessageResponse("User registered successfully!"));
}
```

Demo

What's Next

- Friends Connection - Allowing a User to invite their support networks to see their Survey Results, Occurrence Log and log occurrences on their behalf
- Occurrence Log has graphical representation of data using Google Chart API to aggregate data and find specific patterns in anxiety occurrences
- Make Tara app more responsive and Smartphone friendly