Milestone 1 Report

Karim Alaa Mohey Eldeen	19016172
Louay Magdy Abd El halim	19016195
Mohamed Magdy Elsayed	19016479
Youhanna Yousri Fawzy	19016899
Youssef Saber Saaid	19016924
Youssef Magdy Helmy	19016937

- Functional Requirements:

1. User Signup

Function	Creates an account for user
Description	Creates an account for user after taking some data after doing the following steps: - Validating user data - Checking if there is a duplicate user account - Creating user record in the DB
Inputs	- User first and last names - Password following some rules - Valid email - National ID - Birth date - Last date of donation - If the user has any diseases preventing blood donation - Blood type
Source	Signup form in the frontend, text fields are used to input the above fields
Outputs	- Creating a record for the user in the DB - Creating an account for the user and directing the user to start page
Action	- Validating user data such as: • Names should not contain digits • National ID fields are valid, and date of birth is extracted from it • Password must contain upper, lower cases, digits, _ and must not contain any other types of characters - If input is valid, the DB state is checked for a duplicate account and if there is no duplicate account, a record is created in the DB. Also the user is routed to the start page again
Requirements	DB Connection established and has user table
Side-effects	Adding new record to the DB state and creating new account if input is valid, and there is not another similar account

2. User login

Function	Logs the user into the application.
Description	Logs the user into the application after validation his credentials, returns his saved information and a jwt token.
Inputs	User email and password.
Source	Login form in the frontend, text fields are used to input the email and password.
Outputs	User information, jwt token.
Action	The application takes the email and password of the user, checks if a user exists in the database and the password matches. Returns user information and jwt token, for further request, if the credentials were valid.
Requirements	Database connection established and has User table.
Side Effects	User has to use jwt token for further requests.

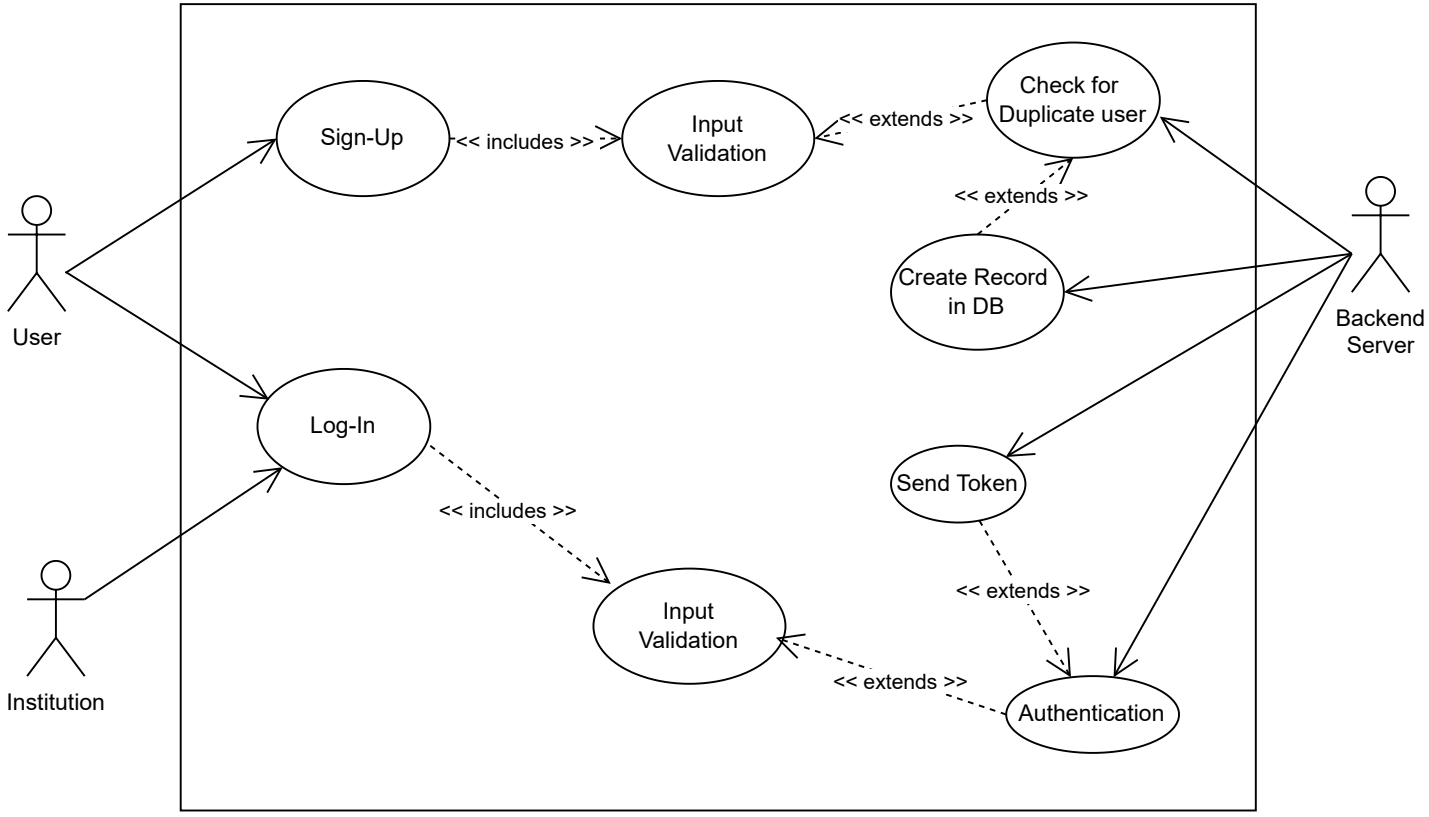
3. Institution login

Function	Logs the institution into the application
Description	The application takes the email and password of the institution, validates it and returns a jwt token, for further request, if the credentials were valid.
Inputs	User email and password
Source	Login form in the frontend, text fields are used to input the email and password.
Outputs	institution information, jwt token.
Destination	The application takes the email and password of the user, checks if a user exists in the database and the password matches. Returns user information and jwt token, for further request, if the credentials were valid.
Action	The application takes the email and password of the institution, checks if a user exists in the database and the password matches. Returns institution information and jwt token, for further request, if the credentials were valid.
Side Effects	institution has to use jwt token for further requests.

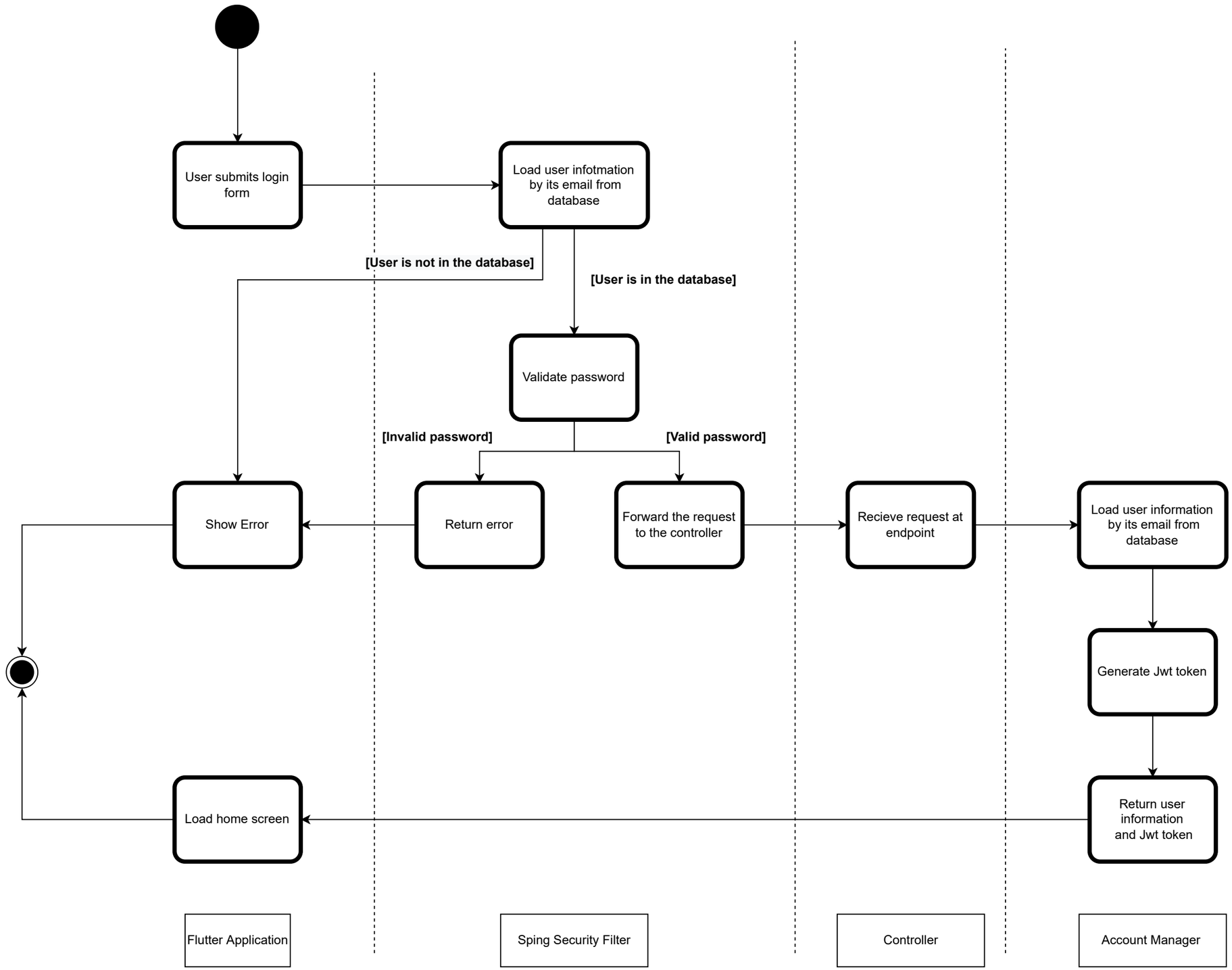
- Non-Functional Requirements:

  Security
  - Verifying some email once he signed up
  - Receiving some token once he signed in
  - Through Password Hashing before saving into the DB using Spring Security

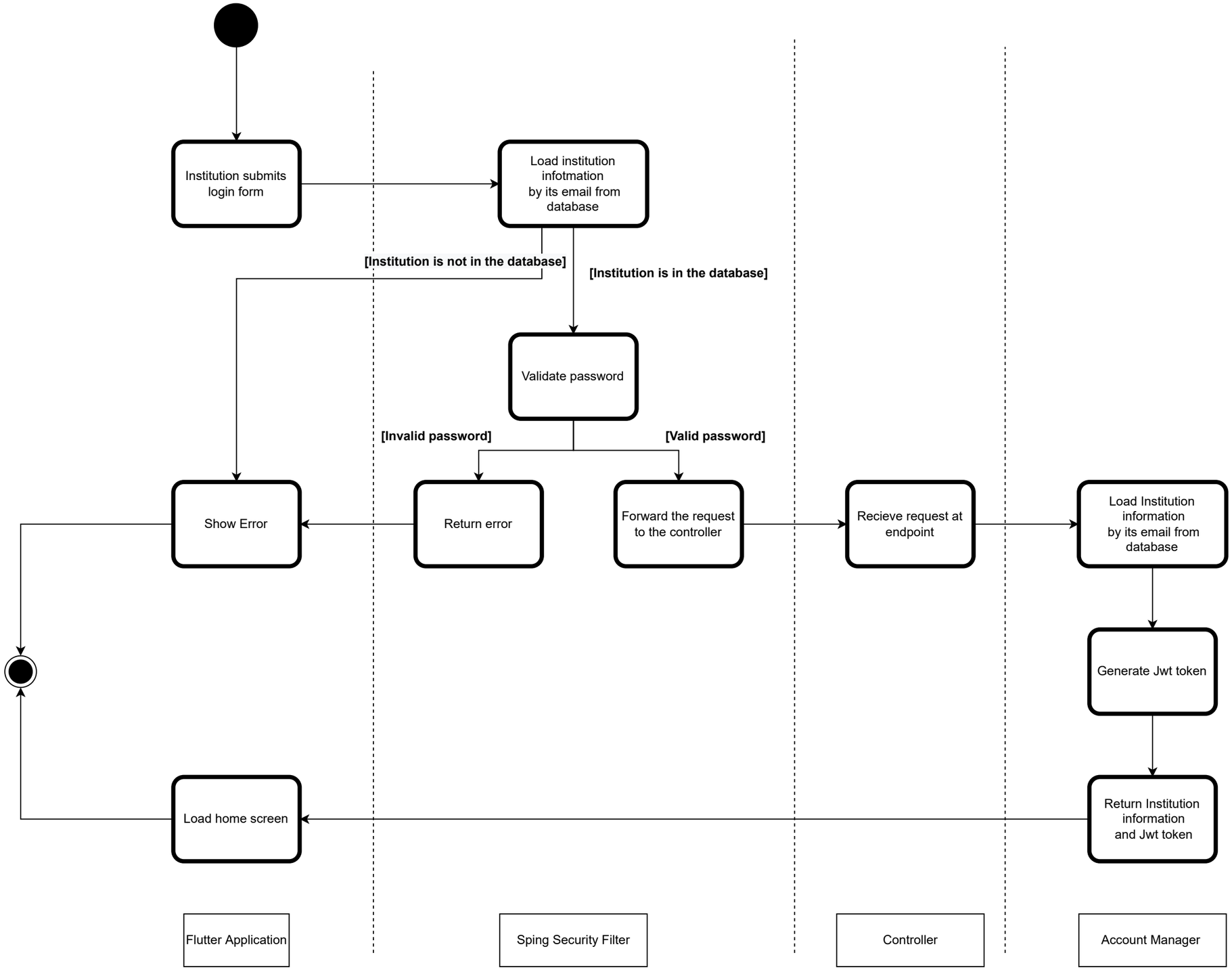
Detailed Use case diagram:



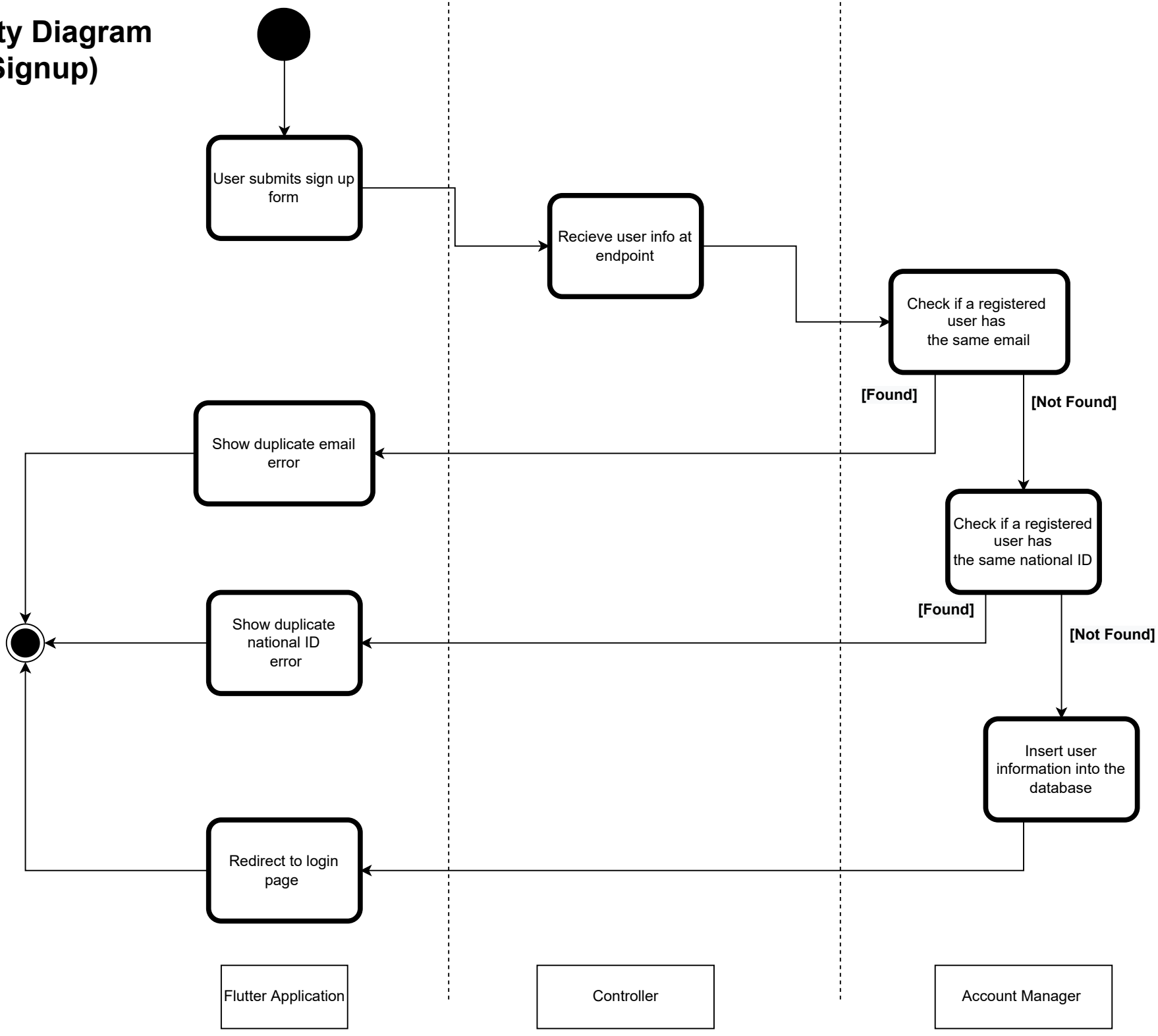
Activity Diagram (User Login)



Activity Diagram (Institution Login)



Activity Diagram (Signup)



Account Managment Component Class Diagram

The current design is very similar to the proposed one except for handling JSON object, as Spring Framework provides JSON Serialization and Deserialization.
The only addition is the Security and Authorization classes.

A security pipeline that filters all requests (except signup) and check for authorization

SecurityConfiguration

- userDetailsService: UserDetailsService
- insDetailsService: UserDetailsService
- rsaKeys: RsaKeyProperties

+ passwordEncoder(): PasswordEncoder
+ webSecurityCustomizer(): WebSecurityCustomizer
+ userFilterChain(httpSecurity: HttpSecurity): SecurityFilterChain
+ instFilterChain(httpSecurity: HttpSecurity): SecurityFilterChain
~ jwtEncoder(): JwtEncoder
~ jwtDecoder(): JwtDecoder
- userAuthenticationManager(http: HttpSecurity): AuthenticationManager
- instAuthenticationManager(http: HttpSecurity): AuthenticationManager

Conceptual class that relies on spring to validate all constraints. Throws exception which is handled in controller

Validator

+ validate(Agent): boolean

UserValidator

- constraints

+ validate(User: user): boolean

InstitutionValidator

- constraints

+ validate(Institution: institution): boolean

AccountController

- accountManagerService: AccountManagerService

+ AccountController(accountManagerService: AccountManagerService)
+ signUp(User: user): SignUpResponse
+ signUp(Institution: insititution): SignUpResponse
+ signInUser(Authentication: credentials): SignInResponse
+ signInInstitution(Authentication: credentials): SignInResponse
+ logOutUser(Authentication: credentials): void
+ logOutInstitution(Authentication: credentials): void
+ handleSIgnupException(exception: SignupException): SignupResponse
+ handleIncorrectFormatException(exception: NotReadableException): SignupRespons

uses

Parser

+ signUpValidate(Agent): boolean
+ signInValidate(Agent): boolean

Conceptual class that relies on spring to validate all constraints. Throws exception which is handled in controller

UserParser

- json: JSON

+ parse(agent: string): User

InstitutionParser

- json: JSON

+ parse(agent: string): Institution

<<Interface>>
AccountManagerService

+ signUp(User: user): boolean
+ signUp(Institution: insititution): boolean
+ signInUser(Authentication: credentials): SignInResponse
+ signInInstitution(Authentication: credentials): SignInResponse
+ logOutUser(Authentication: credentials): void
+ logOutInstitution(Authentication: credentials): void

AccountManagerServiceImp

- userDAO: UserDAO
- institutionDAO: Registerer
- tokenUtil: TokenUtil

+ signUp(User: user): boolean
+ signUp(Institution: insititution): boolean
+ signInUser(Authentication: credentials): SignInResponse
+ signInInstitution(Authentication: credentials): SignInResponse
+ logOutUser(Authentication: credentials): void
+ logOutInstitution(Authentication: credentials): void

TokenUtil

- encoder: JwtEncoder

+ generateToken(authentication: Prinicpal): String

UserDAO

- userRepo: UserRepository

+ saveUser(newUser: User): boolean
+ findUserByEmail(email: String): User
+ findUserByNationalID(String: nationalID): User
+ isUserAndPasswordMatching(email:String, password: String): boolean
+ getUsersByBloodType(String bloodType): List<User>
+ loadUserByUsername(username: String): UserDetails

InstitutionDAO

- instRepo: InstitutionRepository

+ saveInstitution(newInstitution: Institution): boolean
+ findInstitutionByEmail(email: String): Institution
+ isUserAndPasswordMatching(email:String, password: String): boolean
+ haveBloodPackets(String bloodType): List<Institution>
+ loadUserByUsername(username: String): UserDetails

DB

User

- ID: number
- firstName: string
- lastName: string
- email: string
- password: string
- national ID: string
- Blood type: String
- hasDiseases: boolean
- lastTimeDonated: Date

+ Setters and Getters
+ User(email, password)
+ User(all attributes except ID)

Institution

- ID: number
- name: string
- email: string
- password: String
- name: String
- literalLocation: String
- GeoLocation: (number, number)
- workingHours: number
- bloodBagsCount: arrayOfNumbers

+ Setters and Getters
+ Institution(email, Password)
+ Institution(all attributes except ID)

<<Interface>>
springframework.security.core.UserDetails

+ getAuthorities(): Collection<GrantedAuthority>
+ getPassword(): String
+ getUsername(): String
+ isAccountNonExpired(): boolean
+ isAccountNonLocked(): boolean
+ isCredentialsNonExpired(): booleans
+ isEnabled(): boolean

UserAuthentication

- user: User

+ UserAuthentication(user: User)
+ getAuthorities(): Collection<GrantedAuthority>
+ getCredentials(): Object
+ getPrincipal(): Object
+ isAuthenticated(): boolean
+ setAuthenticated(isAuthenticated: boolean): void

InstitutionAuthentication

- institution: Institution

+ UserAuthentication(user: User)
+ getAuthorities(): Collection<GrantedAuthority>
+ getCredentials(): Object
+ getPrincipal(): Object
+ isAuthenticated(): boolean
+ setAuthenticated(isAuthenticated: boolean): void