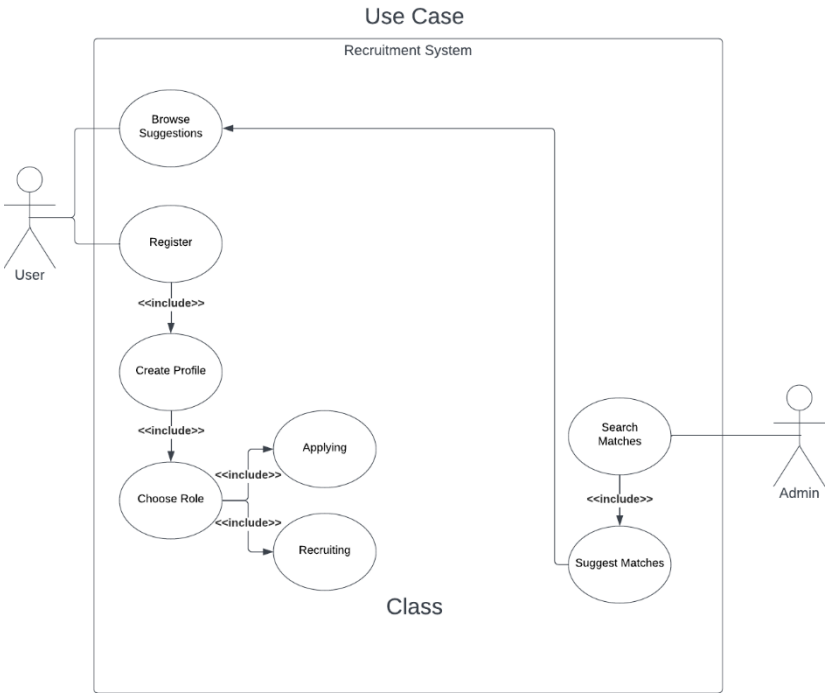
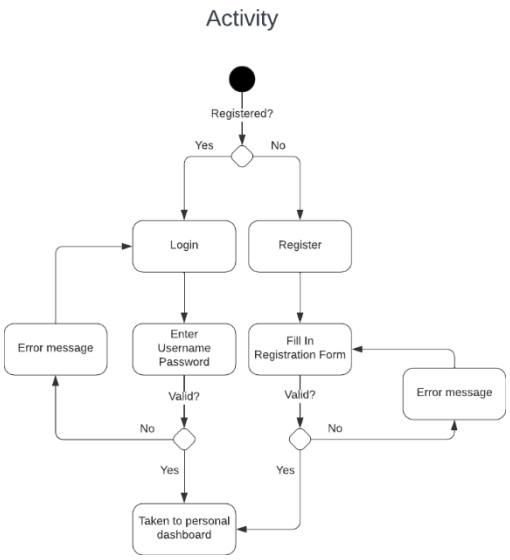
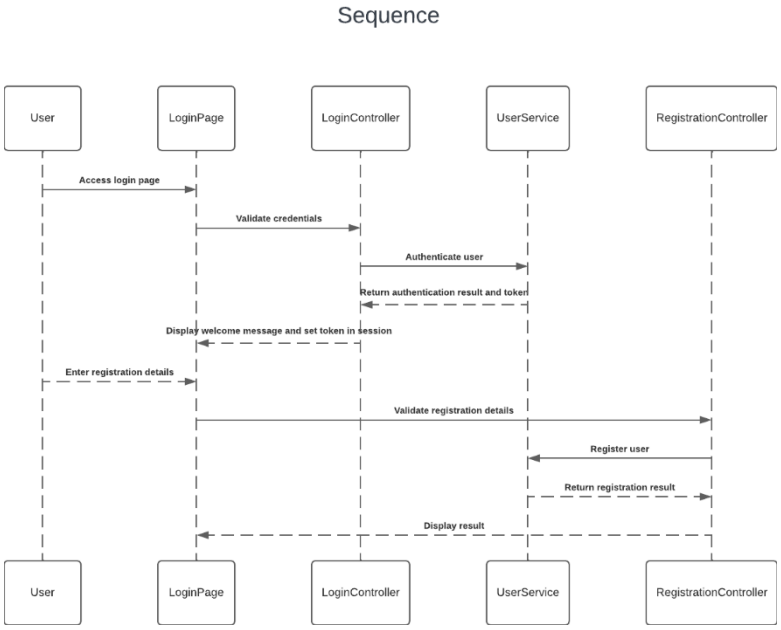
UML-diagram och flowchart



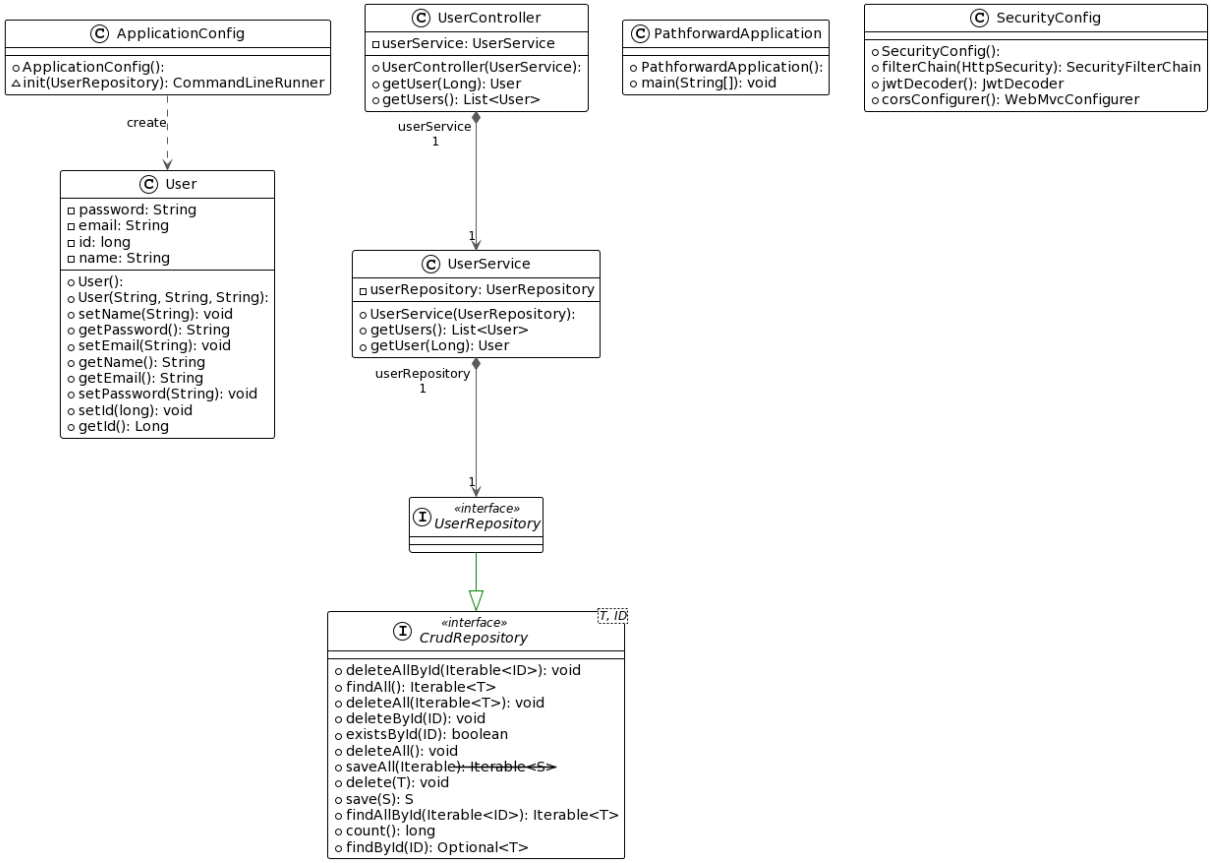
Figur 1: Use Case-diagram med användaren till vänster. Användaren kan: bläddra bland förslag som kommer från handläggaren; registrera sig, skapa en profil, välja en roll – ”applying” eller ”recruiting” (även ”Admin” borde vara med). Admin/handläggare kan söka efter och föreslå matchningar.



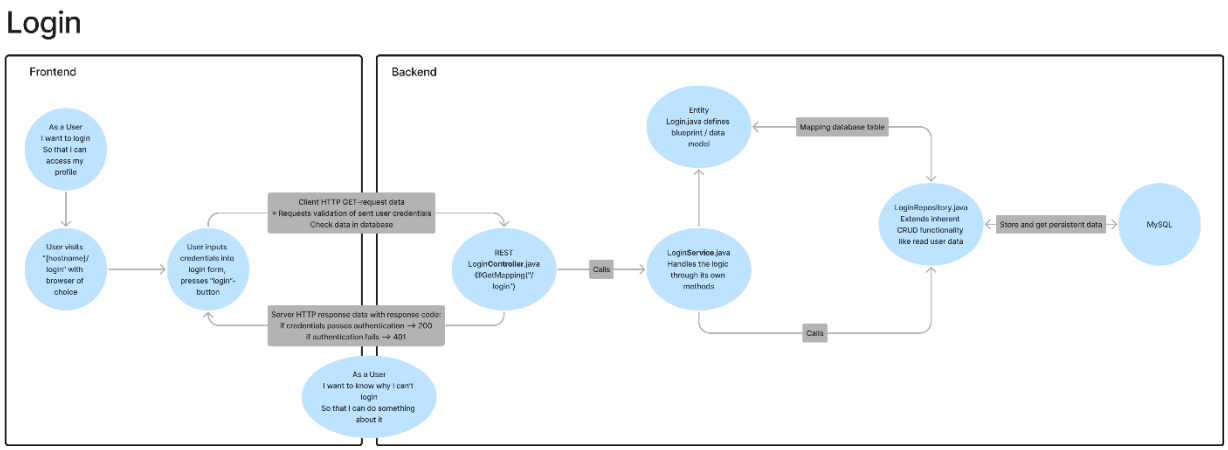
Figur 2: Activity-diagram med utgångsläge längst upp i diagrammet, som sedan följer olika vägar beroende på om användaren är registrerad eller inte. Senare valideras även input från användaren och felmeddelande kan skrivas ut.



Figur 3: Sequence-diagram över login-/registrerings- och autentiseringsflödet. Användaren går in på loginsidan som skickar userdata till LoginController, som skickar vidare en förfrågan om autentisering till UserService. Därefter skickar UserService ett svar om autentiseringsresultatet tillsammans med en token (om det har fungerat, där även felhantering borde ingå). LoginController skickar ett meddelande till frontend och token sätts (i en HttpOnly-cookie förslagsvis). Användaren registrerar sig (egentligen innan login givetvis), skickar data till RegistrationController, som skickar vidare själva logiken för registrering till UserService. Resultatet skickas till Controllern, som skickar svar till frontend. Man skulle även kunna inkludera databas-, crud-interface- och entitets-lager, med UserRepository som använder sig av entitetens datamodell.



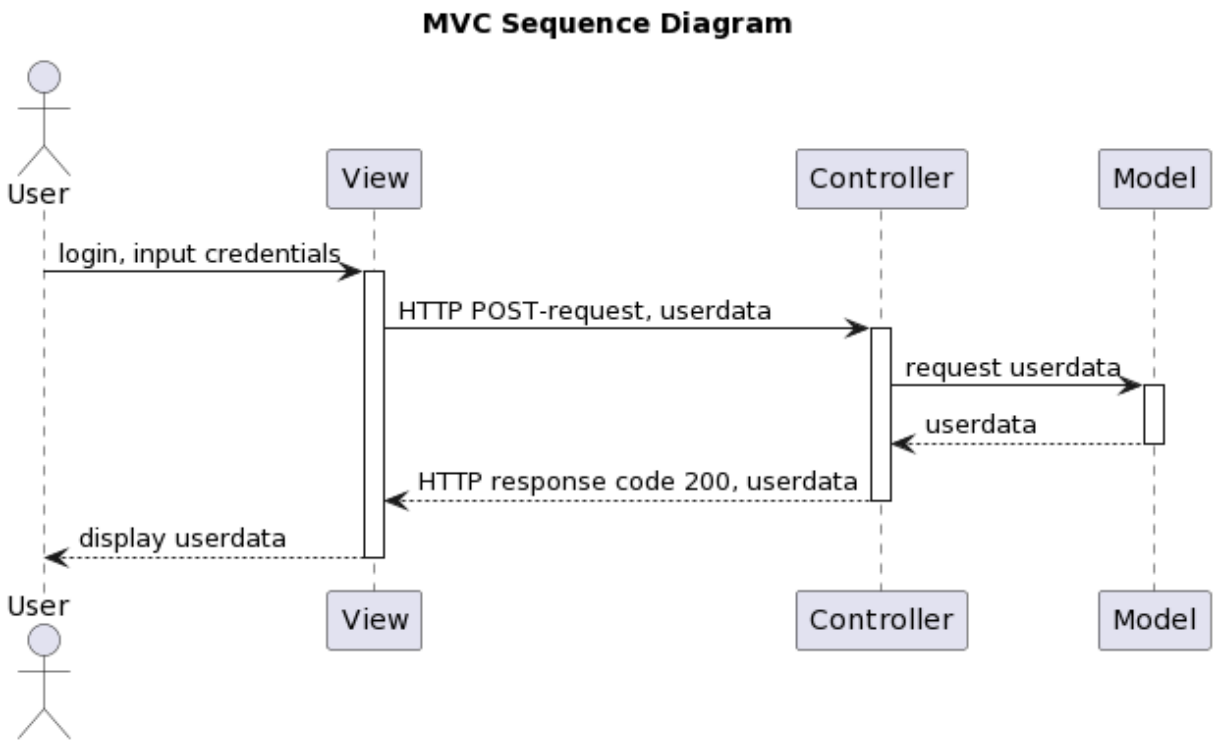
Figur 4: Class-diagram med User-klass som skapas av ApplicationConfig. UserService har ett 1:1 aggregationsförhållande till UserController, det vill säga att Controllern använder sig av en instantiering av Service-klassen, och de båda har som klasser fristående ”lifelines”. UserService har i sin tur ett likadant förhållande till UserRepository som dessutom extendar CrudRepository. PathforwardApplication samt SecurityConfig är fristående klasser som är med för att visa dess egenskaper (i första rutan efter klassnamnet), och metoder (nedanför första rutan)..



Figur 5: Flowchart för hur ett autentiseringsförlopp skulle kunna fungera mellan frontend, backend och databas, med user stories för användarcentrering. En användare (A1 i det följande) vill logga in sannolikt för att komma åt sin profilsida, och agerar utifrån den önskan. A1 fyller i Login-form, och trycker på knappen ”Login” → POST-request skickas via Axios till Controller på backend, som sedan skickar HTTP-svar med antingen 200: OK, eller 401: Unauthorized.

För att ännu tydligare visualisera det som förklaras i resultatet används ett sequence-diagram skapat med PlantUML[1], utifrån en Model-View-Controller-design (MVC)[2]:

[1] PlantUML Web Server
[2] Model–view–controller - Wikipedia



Figur 6:Sequence-diagram över MVC-design.

MVC handlar om ett separationstänk där Model kan sägas representera data och dess logik; View gränssnittet för användaren; och Controllern fungerar som en fördelare som tar emot användar-input via HTTP-metoder och returnerar svar med HTTP-svarskoder. I det här fallet skickas en POST-request med userdata till Controllern, som deserialiserar (konverterar) JSON till ett Java object och fördelar den vidare till Model-lagret innehållande Service, Repository, databas, för eventuell hantering av datan. Den data som efterfrågas skickas tillbaka till Controller, som serialiserar till JSON och View kan ta emot detta och i sin tur serialisera JSON till JavaScript-objekt som kan visas för användaren. Detta är ett förenklat flöde som inte tar hänsyn till behovet av validering, autentisering och att enbart skicka nödvändiga data med hjälp av Data Transfer Objects för att något annat är sämre både ur säkerhets- och effektivitetssynpunkt[3].

[3] The DTO Pattern (Data Transfer Object) | Baeldung