Complex Method

Smell Location:

File: AuthProvider.java

Class: AuthProvider

Method: authenticate

Line Number: 45

Refactoring Technique: Split the authenticate method into smaller methods.

Commit hash: complex method authenticate solved in the AuthProvider.java.

Before Refactoring:

Tests Passed: 35

Tests Failed: 0

After Refactoring:

Tests Passed: 35

Tests Failed: 0

**Complex Problem Before**:

A screen shot of a computer program

Description automatically generatedA computer screen shot of text

Description automatically generated

**After :**

**A computer screen shot of a program code

Description automatically generatedA computer screen shot of text

Description automatically generated**

**LONG METHOD**:

Smell Location:

File: UserDto.java

Class: UserDto

Method: userToUserDto

Line Number: 77

Refactoring Technique: Split the userToUserDto method into smaller methods.

Commit hash: solved the long method code smell in UserDto

*Before Refactoring:*

Tests Passed: 35

Tests Failed: 0

*After Refactoring:*

Tests Passed: 35

Tests Failed: 0

BEFORE

*public* *static* UserDto *userToUserDto*(DtoBuilder builder)

*throws* InvalidKeyException, IllegalBlockSizeException, BadPaddingException, NoSuchAlgorithmException,

            NoSuchPaddingException, InvalidAlgorithmParameterException, UnsupportedEncodingException, AlovoaException {

        User user = builder.*user*;

        User currentUser = builder.*currentUser*;

        UserService userService = builder.*userService*;

        boolean ignoreIntention = builder.*ignoreIntention*;

*final* UUID uuid;

*if* (user == null) {

*return* null;

        }

        UserDto dto = *new* *UserDto*();

*if* (user.*equals*(currentUser)) {

            dto.*setEmail*(user.*getEmail*());

            dto.*setLocationLatitude*(user.*getLocationLatitude*());

            dto.*setLocationLongitude*(user.*getLocationLongitude*());

            UserSettings settings = user.*getUserSettings*();

            dto.*setUserSettings*(settings);

        }

        uuid = Tools.*getUserUUID*(user, userService);

        dto.*setUuid*(uuid);

*if* (user.*getDates*() != null) {

            dto.*setAge*(Tools.*calcUserAge*(user));

        }

*if* (user.*getLocationLatitude*() != null) {

            dto.*setHasLocation*(true);

        }

        dto.*setDescription*(user.*getDescription*());

        dto.*setFirstName*(user.*getFirstName*());

        dto.*setGender*(user.*getGender*());

*if* (user.*getVerificationPicture*() != null) {

            dto.*setVerificationPicture*(UserDtoVerificationPicture.*map*(user, currentUser, userService, user.*getVerificationPicture*().*getUuid*()));

        }

        dto.*setCountry*(Tools.*getCountryEmoji*(user.*getCountry*()));

*if* (currentUser.*isShowZodiac*()) {

            dto.*setZodiac*(*getUserZodiac*(user));

        }

        dto.*setShowZodiac*(user.*isShowZodiac*());

        dto.*setUnits*(user.*getUnits*());

        dto.*setMiscInfos*(user.*getMiscInfos*());

        dto.*setPreferedGenders*(user.*getPreferedGenders*());

        dto.*setPreferedMinAge*(user.*getPreferedMinAge*());

        dto.*setPreferedMaxAge*(user.*getPreferedMaxAge*());

*if* (dto.*getPreferedMinAge*() < Tools.*AGE\_LEGAL* && dto.*getAge*() >= Tools.*AGE\_LEGAL*) {

            dto.*setPreferedMinAge*(Tools.*AGE\_LEGAL*);

        }

        dto.*setImages*(UserImageDto.*buildFromUserImages*(user, userService));

        dto.*setGender*(user.*getGender*());

        dto.*setIntention*(user.*getIntention*());

*if* (user.*getProfilePicture*() != null) {

            UUID picUuid = Tools.*getProfilePictureUUID*(user.*getProfilePicture*(), userService);

            dto.*setProfilePicture*(UserProfilePicture.*getPublicUrl*(userService.*getDomain*(), picUuid));

        }

        dto.*setTotalDonations*(user.*getTotalDonations*());

        dto.*setNumBlockedByUsers*(user.*getBlockedByUsers*().*size*());

        dto.*setNumReports*(user.*getReportedByUsers*().*size*());

        dto.*setInterests*(user.*getInterests*());

*if* (user.*getAudio*() != null) {

            UUID picUuid = user.*getAudio*().*getUuid*() != null *?* user.*getAudio*().*getUuid*() *:* uuid;

            dto.*setAudio*(userService.*getDomain*() + MediaController.*URL\_REQUEST\_MAPPING* +

                    MediaController.*URL\_AUDIO* + picUuid);

        }

        dto.*setHasAudio*(user.*getAudio*() != null);

        dto.*setNumberReferred*(user.*getNumberReferred*());

        dto.*setPrompts*(user.*getPrompts*());

*if* (!user.*isAdmin*()) {

            LocalDateTime now = LocalDateTime.*now*();

            LocalDateTime activeDateTime = Tools.*dateToLocalDateTime*(user.*getDates*().*getActiveDate*());

*if* (activeDateTime.*isAfter*(now.*minusMinutes*(LA\_STATE\_ACTIVE\_1))) {

                dto.*setLastActiveState*(1);

            } *else* *if* (activeDateTime.*isAfter*(now.*minusDays*(LA\_STATE\_ACTIVE\_2))) {

                dto.*setLastActiveState*(2);

            } *else* *if* (activeDateTime.*isAfter*(now.*minusDays*(LA\_STATE\_ACTIVE\_3))) {

                dto.*setLastActiveState*(3);

            } *else* *if* (activeDateTime.*isAfter*(now.*minusDays*(LA\_STATE\_ACTIVE\_4))) {

                dto.*setLastActiveState*(4);

            }

        }

*if* (!user.*equals*(currentUser)) {

*if* (currentUser.*getBlockedUsers*() != null) {

                dto.*blockedByCurrentUser* = currentUser.*getBlockedUsers*().*stream*()

                        .*filter*(o -> o.*getUserTo*() != null)

                        .*anyMatch*(o -> Objects.*equals*(o.*getUserTo*().*getId*(), user.*getId*()));

            }

*if* (currentUser.*getReported*() != null) {

                dto.*reportedByCurrentUser* = currentUser.*getReported*().*stream*()

                        .*filter*(o -> o.*getUserTo*() != null)

                        .*anyMatch*(o -> Objects.*equals*(o.*getUserTo*().*getId*(), user.*getId*()));

            }

*if* (user.*getLikes*() != null) {

                dto.*likesCurrentUser* = user.*getLikes*().*stream*()

                        .*filter*(o -> o.*getUserTo*() != null)

                        .*anyMatch*(o -> Objects.*equals*(o.*getUserTo*().*getId*(), currentUser.*getId*()));

            }

*if* (currentUser.*getLikes*() != null) {

                dto.*likedByCurrentUser* = currentUser.*getLikes*().*stream*()

                        .*filter*(o -> o.*getUserTo*() != null)

                        .*anyMatch*(o -> Objects.*equals*(o.*getUserTo*().*getId*(), user.*getId*()));

            }

*if* (currentUser.*getHiddenUsers*() != null) {

                dto.*hiddenByCurrentUser* = currentUser.*getHiddenUsers*().*stream*()

                        .*filter*(o -> o.*getUserTo*() != null)

                        .*anyMatch*(o -> Objects.*equals*(o.*getUserTo*().*getId*(), user.*getId*()));

            }

            List<UserInterest> commonInterests = *new* ArrayList<>();

*for* (int i = 0; i < currentUser.*getInterests*().*size*(); i++) {

                UserInterest interest = currentUser.*getInterests*().*get*(i);

*if* (user.*getInterests*().*contains*(interest)) {

                    commonInterests.*add*(interest);

                }

            }

            dto.*setCommonInterests*(commonInterests);

            int dist = 99999;

*if* (!currentUser.*isAdmin*()) {

                dist = Tools.*getDistanceToUser*(user, currentUser);

*if* (currentUser.*getUnits*() == User.*UNIT\_IMPERIAL*) {

                    dist = (int) (dist \* MILES\_TO\_KM);

                }

            }

            dto.*setDistanceToUser*(dist);

        }

        dto.*setCompatible*(Tools.*usersCompatible*(currentUser, user, ignoreIntention));

*return* dto;

    }

After:

*public* *static* UserDto *userToUserDto*(DtoBuilder builder)

*throws* InvalidKeyException, IllegalBlockSizeException, BadPaddingException, NoSuchAlgorithmException,

        NoSuchPaddingException, InvalidAlgorithmParameterException, UnsupportedEncodingException, AlovoaException {

*if* (builder.*user* == null) {

*return* null;

    }

    UserDto dto = *new* *UserDto*();

*if* (builder.*user*.*equals*(builder.*currentUser*)) {

*populateCurrentUserDetails*(dto, builder.*user*);

    }

    UUID uuid = Tools.*getUserUUID*(builder.*user*, builder.*userService*);

    dto.*setUuid*(uuid);

*populateUserBasicDetails*(dto, builder.*user*, builder.*userService*, builder.*currentUser*, uuid);

*populateUserCompatibilityDetails*(dto, builder.*user*, builder.*currentUser*, builder.*userService*, builder.*ignoreIntention*);

*return* dto;

}

*private* *static* void *populateCurrentUserDetails*(UserDto dto, User user) {

    dto.*setEmail*(user.*getEmail*());

    dto.*setLocationLatitude*(user.*getLocationLatitude*());

    dto.*setLocationLongitude*(user.*getLocationLongitude*());

    dto.*setUserSettings*(user.*getUserSettings*());

}

*private* *static* void *populateUserBasicDetails*(UserDto dto, User user, UserService userService, User currentUser, UUID uuid)

*throws* InvalidKeyException, IllegalBlockSizeException, BadPaddingException, NoSuchAlgorithmException,

        NoSuchPaddingException, InvalidAlgorithmParameterException, UnsupportedEncodingException {

*if* (user.*getDates*() != null) {

        dto.*setAge*(Tools.*calcUserAge*(user));

    }

    dto.*setHasLocation*(user.*getLocationLatitude*() != null);

    dto.*setDescription*(user.*getDescription*());

    dto.*setFirstName*(user.*getFirstName*());

    dto.*setGender*(user.*getGender*());

    dto.*setVerificationPicture*(UserDtoVerificationPicture.*map*(user, currentUser, userService, user.*getVerificationPicture*().*getUuid*()));

    dto.*setCountry*(Tools.*getCountryEmoji*(user.*getCountry*()));

    dto.*setShowZodiac*(user.*isShowZodiac*());

    dto.*setZodiac*(currentUser.*isShowZodiac*() *?* *getUserZodiac*(user) *:* null);

    dto.*setUnits*(user.*getUnits*());

    dto.*setMiscInfos*(user.*getMiscInfos*());

    dto.*setPreferedGenders*(user.*getPreferedGenders*());

    dto.*setPreferedMinAge*(Math.*max*(user.*getPreferedMinAge*(), (dto.*getAge*() >= Tools.*AGE\_LEGAL* *?* Tools.*AGE\_LEGAL* *:* dto.*getPreferedMinAge*())));

    dto.*setPreferedMaxAge*(user.*getPreferedMaxAge*());

    dto.*setImages*(UserImageDto.*buildFromUserImages*(user, userService));

    dto.*setProfilePicture*(*getUserProfilePicture*(user, userService));

    dto.*setTotalDonations*(user.*getTotalDonations*());

    dto.*setNumBlockedByUsers*(user.*getBlockedByUsers*().*size*());

    dto.*setNumReports*(user.*getReportedByUsers*().*size*());

    dto.*setInterests*(user.*getInterests*());

    dto.*setAudio*(user.*getAudio*() != null *?* userService.*getDomain*() + MediaController.*URL\_REQUEST\_MAPPING* + MediaController.*URL\_AUDIO* + (user.*getAudio*().*getUuid*() != null *?* user.*getAudio*().*getUuid*() *:* uuid) *:* null);

    dto.*setHasAudio*(user.*getAudio*() != null);

    dto.*setNumberReferred*(user.*getNumberReferred*());

    dto.*setPrompts*(user.*getPrompts*());

}

*private* *static* String *getUserProfilePicture*(User user, UserService userService) *throws* UnsupportedEncodingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorithmParameterException {

*return* user.*getProfilePicture*() != null *?* UserProfilePicture.*getPublicUrl*(userService.*getDomain*(), Tools.*getProfilePictureUUID*(user.*getProfilePicture*(), userService)) *:* null;

}

*private* *static* void *populateUserCompatibilityDetails*(UserDto dto, User user, User currentUser, UserService userService, boolean ignoreIntention) {

*if* (!user.*equals*(currentUser)) {

        dto.*setBlockedByCurrentUser*(*isUserBlockedByCurrentUser*(currentUser, user));

        dto.*setReportedByCurrentUser*(*isUserReportedByCurrentUser*(currentUser, user));

        dto.*setLikesCurrentUser*(*isUserLikedByCurrentUser*(user, currentUser));

        dto.*setLikedByCurrentUser*(*isUserLikedByCurrentUser*(currentUser, user));

        dto.*setHiddenByCurrentUser*(*isUserHiddenByCurrentUser*(currentUser, user));

        dto.*setCommonInterests*(*getCommonInterests*(currentUser, user));

        dto.*setDistanceToUser*(*getDistanceToUser*(user, currentUser, userService));

    }

    dto.*setCompatible*(Tools.*usersCompatible*(currentUser, user, ignoreIntention));

*if* (!user.*isAdmin*()) {

        dto.*setLastActiveState*(*getLastActiveState*(user));

    }

}

*private* *static* boolean *isUserBlockedByCurrentUser*(User currentUser, User user) {

*return* currentUser.*getBlockedUsers*().*stream*().*filter*(o -> o.*getUserTo*() != null).*anyMatch*(o -> Objects.*equals*(o.*getUserTo*().*getId*(), user.*getId*()));

}

*private* *static* boolean *isUserReportedByCurrentUser*(User currentUser, User user) {

*return* currentUser.*getReported*().*stream*().*filter*(o -> o.*getUserTo*() != null).*anyMatch*(o -> Objects.*equals*(o.*getUserTo*().*getId*(), user.*getId*()));

}

*private* *static* boolean *isUserLikedByCurrentUser*(User user, User currentUser) {

*return* user.*getLikes*().*stream*().*filter*(o -> o.*getUserTo*() != null).*anyMatch*(o -> Objects.*equals*(o.*getUserTo*().*getId*(), currentUser.*getId*()));

}

*private* *static* boolean *isUserHiddenByCurrentUser*(User currentUser, User user) {

*return* currentUser.*getHiddenUsers*().*stream*().*filter*(o -> o.*getUserTo*() != null).*anyMatch*(o -> Objects.*equals*(o.*getUserTo*().*getId*(), user.*getId*()));

}

*private* *static* List<UserInterest> *getCommonInterests*(User currentUser, User user) {

    List<UserInterest> commonInterests = *new* ArrayList<>();

*for* (UserInterest interest *:* currentUser.*getInterests*()) {

*if* (user.*getInterests*().*contains*(interest)) {

            commonInterests.*add*(interest);

        }

    }

*return* commonInterests;

}

*private* *static* int *getDistanceToUser*(User user, User currentUser, UserService userService) {

    int dist = 99999;

*if* (!currentUser.*isAdmin*()) {

        dist = Tools.*getDistanceToUser*(user, currentUser);

*if* (currentUser.*getUnits*() == User.*UNIT\_IMPERIAL*) {

            dist = (int) (dist \* MILES\_TO\_KM);

        }

    }

*return* dist;

}

*private* *static* int *getLastActiveState*(User user) {

    LocalDateTime now = LocalDateTime.*now*();

    LocalDateTime activeDateTime = Tools.*dateToLocalDateTime*(user.*getDates*().*getActiveDate*());

*if* (activeDateTime.*isAfter*(now.*minusMinutes*(LA\_STATE\_ACTIVE\_1))) {

*return* 1;

    } *else* *if* (activeDateTime.*isAfter*(now.*minusDays*(LA\_STATE\_ACTIVE\_2))) {

*return* 2;

    } *else* *if* (activeDateTime.*isAfter*(now.*minusDays*(LA\_STATE\_ACTIVE\_3))) {

*return* 3;

    } *else* *if* (activeDateTime.*isAfter*(now.*minusDays*(LA\_STATE\_ACTIVE\_4))) {

*return* 4;

    } *else* {

*return* 0;

    }

}

Long Identifier:

Smell Location:

File: MediaService.java

Class: MediaService

Method: getVerificationPicture

Line Number: 57

Refactoring Technique: gave smaller names to the variables in the code

Commit hash: solved long identifier code smell by giving them short names

*Before Refactoring:*

Tests Passed: 35

Tests Failed: 0

*After Refactoring:*

Tests Passed: 35

Tests Failed: 0

Before:

A screen shot of a computer program

Description automatically generated

After:

A screenshot of a computer program

Description automatically generated