Chapter 2: User-Facing Backend

Ticket: User Management

Back to the Question

Here are the possible implementations for the methods required by this ticket:

COPY

```
public class UserDao extends AbstractMFlixDao {

  private final MongoCollection<User> usersCollection;
  private final MongoCollection<Session> sessionsCollection;
  private final Logger log;

  @Autowired
  public UserDao(
    MongoClient mongoClient,
@Value("${spring.mongodb.database}") String databaseName) {
    super(mongoClient, databaseName);
    CodecRegistry pojoCodecRegistry =
        fromRegistries(
          MongoClientSettings.getDefaultCodecRegistry(),

fromProviders(PojoCodecProvider.builder().automatic(true).build
()));

    usersCollection = db.getCollection("users",
User.class).withCodecRegistry(pojoCodecRegistry);
    log = LoggerFactory.getLogger(this.getClass());
    sessionsCollection =
      db.getCollection("sessions",
```

```
Session.class).withCodecRegistry(pojoCodecRegistry);
    }
  /*
  * @param user - User object to be added
  * @return True if successful, false otherwise.
  */
  public boolean addUser(User user) {

usersCollection.withWriteConcern(WriteConcern.MAJORITY).insertO
ne(user);
      return true;
    }

  /*
  * Creates session using userId and jwt token.
  *
  * @param userId - user string identifier
  * @param jwt - jwt string token
  * @return true if successful
  */
  public boolean createUserSession(String userId, String jwt) {
    Bson updateFilter = new Document("user_id", userId);
    Bson setUpdate = Updates.set("jwt", jwt);
    UpdateOptions options = new UpdateOptions().upsert(true);
    sessionsCollection.updateOne(updateFilter, setUpdate,
options);
    return true;
    }

  /*
  * Returns the User object matching the an email string value.
  *
  * @param email - email string to be matched.
  * @return User object or null.
  */
  public User getUser(String email) {
    return usersCollection.find(new Document("email",
email)).limit(1).first();
  }

  /*
  * Given the userId, returns a Session object.
  *
  * @param userId - user string identifier.
```

```
  * @return Session object or null.
  */
  public Session getUserSession(String userId) {
    return sessionsCollection.find(new Document("user_id",
userId)).limit(1).first();
    }

  public boolean deleteUserSessions(String userId) {
    Document sessionDeleteFilter = new Document("user_id",
userId);
    DeleteResult res =
sessionsCollection.deleteOne(sessionDeleteFilter);
    if (res.getDeletedCount() < 1) {
      log.warn("User `{}` could not be found in sessions
collection.", userId);
      }

      return res.wasAcknowledged();
    }

  /**
  * Removes the user document that match the provided email.
  *
  * @param email - of the user to be deleted.
  * @return true if user successfully removed
  */
  public boolean deleteUser(String email) {
  // remove user sessions
    if (deleteUserSessions(email)) {
      Document userDeleteFilter = new Document("email", email);
      DeleteResult res =
usersCollection.deleteOne(userDeleteFilter);

      if (res.getDeletedCount() < 0) {
        log.warn("User with `email` {} not found. Potential
concurrent operation?!");
      }

      return res.wasAcknowledged();
    }
    return false;
  }
}
```