Chapter 4: Resiliency

Ticket: Handling Errors

Here are possible implementations for the methods that required changes.

`MovieDao` possible `validIdValue` implementation:

```
private boolean validIdValue(String movieId){
    try{
        new ObjectId(movieId);
    } catch (IllegalArgumentException e){
        // value cannot be transformed into mongodb ObjectID
        return false;
    }
    return true;
}
```

`CommentDao` possible `addComment` implementation

```
public Comment addComment(Comment comment) {

  if (comment.getId() == null || comment.getId().isEmpty()) {
    throw new IncorrectDaoOperation("Comment objects need to have an
id field set.");
  }

  try {
    commentCollection.insertOne(comment);
    return comment;
  } catch (MongoWriteException e) {
    String errorMessage =
        MessageFormat.format(
            "Error occurred while adding a new Comment `{}`: {}",
comment, e.getMessage());
    throw new IncorrectDaoOperation(errorMessage);
  }
}
```

`CommentDao` possible `updateComment` implementation

COPY

```
public boolean updateComment(String commentId, String text, String
email) {

  Bson filter =
      Filters.and(Filters.eq("email", email), Filters.eq("_id", new
ObjectId(commentId)));
  Bson update = Updates.combine(Updates.set("text", text),
Updates.set("date", new Date()));
  try {

    UpdateResult res = commentCollection.updateOne(filter, update);

    if (res.getMatchedCount() > 0) {

      if (res.getModifiedCount() != 1) {
        log.warn("Comment `{}` text was not updated. Is it the same
text?");
      }

      return true;
    }
    log.error(
        "Could not update comment `{}`. Make sure the comment is
owned by `{}`",
        commentId,
        email);
    return false;

  } catch (MongoWriteException e) {
    String messageError =
        MessageFormat.format(
            "Error occurred while updating comment `{}`: {}",
commentId, e.getMessage());
    throw new IncorrectDaoOperation(messageError);
  }
}
```

`CommentDao` possible `deleteComment` implementation

COPY

```
public boolean deleteComment(String commentId, String email) {

  Bson filter =
      Filters.and(Filters.eq("email", email), Filters.eq("_id", new
ObjectId(commentId)));

  try {
    DeleteResult res = commentCollection.deleteOne(filter);
    if (res.getDeletedCount() != 1) {
      log.warn(
          "Not able to delete comment `{}` for user `{}`. User"
              + " does not own comment or already deleted!",
          commentId,
          email);
      return false;
    }
    return true;
  } catch (MongoWriteException e) {
    String errorMessage =
        MessageFormat.format("Error deleting comment " + "`{}`: {}",
commentId, e);
    throw new IncorrectDaoOperation(errorMessage);
  }
}
```

`UserDao` possible `addUser` implementation

COPY

```
public boolean addUser(User user) {
  try {

usersCollection.withWriteConcern(WriteConcern.MAJORITY).insertOne(us
er);
    return true;

  } catch (MongoWriteException e) {
    log.error(
        "Could not insert `{}` into `users` collection: {}",
user.getEmail(), e.getMessage());
    throw new IncorrectDaoOperation(
        MessageFormat.format("User with email `{0}` already exists",
user.getEmail()));
  }
}
```

UserDao possible createUserSession implementation

```
public boolean createUserSession(String userId, String jwt){
    try{
        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;
    } catch (MongoWriteException e){
      String errorMessage =
      MessageFormat.format(
          "Unable to $set jwt token in sessions collection: {}",
e.getMessage());
      throw new IncorrectDaoOperation(errorMessage, e);
    }
}
```

UserDao possible deleteUser implementation

```
public boolean deleteUser(String email) {
  // remove user sessions
  try {
    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();
    }
  } catch (Exception e) {
    String errorMessage = MessageFormat.format("Issue caught while
trying to " +
          "delete user `{}`: {}",
      email,
      e.getMessage());
    throw new IncorrectDaoOperation(errorMessage);

  }
  return false;
}
```

UserDao possible updateUserPreferences implementation

COPY

```
public boolean updateUserPreferences(String email, Map<String, ?>
userPreferences) {

  // make sure to check if userPreferences are not null. If null,
return false immediately.
  if (userPreferences == null) {
    throw new IncorrectDaoOperation("userPreferences cannot be set
to null");
  }
  // create query filter and update object.
  Bson updateFilter = new Document("email", email);
  Bson updateObject = Updates.set("preferences", userPreferences);
  try {
    // update one document matching email.
    UpdateResult res = usersCollection.updateOne(updateFilter,
updateObject);
    if (res.getModifiedCount() < 1) {
      log.warn(
          "User `{}` was not updated. Trying to re-write the same
`preferences` field: `{}`",
          email,
          userPreferences);
    }
    return true;
  } catch (MongoWriteException e) {
    String errorMessage =
        MessageFormat.format(
            "Issue caught while trying to update user `{}`: {}",
email, e.getMessage());
    throw new IncorrectDaoOperation(errorMessage);
  }
}
```

Proceed to next section