Chapter 2: User-Facing Backend

Ticket: Create/Update Comments

A possible solution would be the following implementation:

COPY

```
/**
 * Adds a new Comment to the collection.
 * The equivalent instruction in the mongo shell would be:
 * <p>
 *     db.comments.insertOne({comment})
 * <p/>
 * @param comment - Comment object.
 * @return Null if the insert fails, otherwise returns the
resulting Comment object.
 */
public Comment addComment(Comment comment){

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


/**
 * Updates the comment text matching commentId and user email.
 * This method would be equivalent to running the following
mongo shell command:
 * <p>
 *     db.comments.update({_id: commentId}, {$set: { "text":
text, date: ISODate() }})
 * <p/>
 * @param commentId - comment id string value.
 * @param text - comment text to be updated.
 * @param email - user email.
 * @return true if successfully updates the comment text.
 */
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())) ;
    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;
}
```

Proceed to next section