# Wednesday: Updating Reviews Model and Tests

## Updating Reviews Model

Now that we have a database we can start saving our reviews to it. We need to update our reviews model.

*app/models.py*

```
from datetime import datetime
........
class Review(db.Model):

    __tablename__ = 'reviews'

    id = db.Column(db.Integer,primary_key = True)
    movie_id = db.Column(db.Integer)
    movie_title = db.Column(db.String)
    image_path = db.Column(db.String)
    movie_review = db.Column(db.String)
    posted = db.Column(db.DateTime,default=datetime.utcnow)
    user_id = db.Column(db.Integer,db.ForeignKey("users.id"))
```

We pass in the `db.Model` class to create a connection to our database. We then add in 6 columns. We use in Python's datetime module to create a timestamp column `posted`. `datetime.utcnow` gets the current time and saves it to our database. We then create Foreign key column where we store the `id` of the user who wrote the review.

*app/models.py*

```
class User(UserMixin,db.Model):
    __tablename__ = 'users'
    .............
    reviews = db.relationship('Review',backref = 'user',lazy = "dynamic")
    ..........
```

We the define the relationship inside our `User` model. We can then create our Review methods

*app/models.py*

```
........
class Review(db.Model):
    __tablename__ = 'reviews'
      ...........
    def save_review(self):
        db.session.add(self)
        db.session.commit()

    @classmethod
    def get_reviews(cls,id):
        reviews = Review.query.filter_by(movie_id=id).all()
        return reviews
```

We create two methods. The `save_review` method will save the instance of the Review model to the session and commit it to the database. The `get_reviews` class method will take in a movie id and retrieve all reviews for that specific movie.

Let us now update our `new_review` view function.

*app/main/views.py*

```
from flask_login import login_required, current_user
#....
@main.route('/movie/review/new/<int:id>', methods = ['GET','POST'])
@login_required
def new_review(id):
    form = ReviewForm()
    movie = get_movie(id)
    if form.validate_on_submit():
        title = form.title.data
        review = form.review.data

        # Updated review instance
        new_review = Review(movie_id=movie.id,movie_title=title,image_path=movie.poster,movie
_review=review,user=current_user)

        # save review method
        new_review.save_review()
        return redirect(url_for('.movie',id = movie.id ))

    title = f'{movie.title} review'
    return render_template('new_review.html',title = title, review_form=form, movie=movie)
```

Here we update how we define our save review instance. Let us also update our database. *manage.py*

```
.......
from app.models import User,Role,Review
.......
```

We first import the Review model to *manage.py* file. We can now create a new migration and upgrade our schema.

## Updating Review Tests

We now need to update the review tests to test changes to our review model. First we need to create a test database and link it to our application. Let us activate our `psql` server and create a test database.

```
(virtual)$ psql
james=# CREATE DATABASE watchlist_test WITH TEMPLATE watchlist
```

Here we create a new database `watchlist_test`. We use `WITH TEMPLATE` to copy the schema of the `watchlist` database so both databases can be identical.

*config.py*

```
........
class TestConfig(Config):
    SQLALCHEMY_DATABASE_URI = 'postgresql+psycopg2://james:password@localhost/watchlist_test'
```

```
class DevConfig(Config):
    SQLALCHEMY_DATABASE_URI = 'postgresql+psycopg2://james:password@localhost/watchlist'
    DEBUG = True

config_options = {
'development':DevConfig,
'production':ProdConfig,
'test':TestConfig
}
```

We then update our configuration files to add a new Config class `TestConfig`. Here we create a new `SQLALCHEMY_DATABASE_URI` to connect to our test database. We then move our development `SQLALCHEMY_DATABASE_URI` to the `DevConfig` file.

Let us now update our tests.

*tests/test_review.py*

```
from app.models import Review,User
from app import db
```

First, we need to import the SQLAlchemy database instance and `User` class.

*tests/test_review.py*

```

#....
def setUp(self):
        self.user_James = User(username = 'James',password = 'potato', email = 'james@ms.com'
)
        self.new_review = Review(movie_id=12345,movie_title='Review for movies',image_path="h
ttps://image.tmdb.org/t/p/w500/jdjdjdjn",movie_review='This movie is the best thing since sli
ced bread',user = self.user_James )
```

First we update our `setUp` method. We create an instance of the `User` and the instance of the `Review` where we pass in the user instance.

*tests/test_review.py*

```
def tearDown(self):
        Review.query.delete()
        User.query.delete()
```

We also update our `tearDown` method. Here we use the `query.delete` method that deletes all elements from the database after every test.

*tests/test_review.py*

```
def test_check_instance_variables(self):
        self.assertEquals(self.new_review.movie_id,12345)
        self.assertEquals(self.new_review.movie_title,'Review for movies')
        self.assertEquals(self.new_review.image_path,"https://image.tmdb.org/t/p/w500/jdjdjdj
n")
        self.assertEquals(self.new_review.movie_review,'This movie is the best thing since sl
iced bread')
        self.assertEquals(self.new_review.user,self.user_James)
```

We then check if the values of variables are correctly being placed.

*tests/test_review.py*

```
    def test_save_review(self):
        self.new_review.save_review()
        self.assertTrue(len(Review.query.all())>0)
```

We then create a test for our save review method. We also query the database to confirm that we actually have data saved.

*tests/test_review.py*

```
def test_get_review_by_id(self):

        self.new_review.save_review()
        got_reviews = Review.get_reviews(12345)
        self.assertTrue(len(got_reviews) == 1)
```

We then test the `get_reviews` class method that we pass in the id of a movie and get a response which is a review for that movie.

*manage.py*

```
app = create_app('test')
```

We then need to update our app instance so that we can use the test database URI.

Now we can run our tests

```
(virtual)$ python3.6 manage.py test
```