Unit Testing with Backbone.js

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    cout << "Hello, world!" << endl;
    return 0;
}
```

A Day in the Life of a MERN Stack Developer

Joe is working really hard to complete the tasks assigned to him.

Recently, Joe has learned Backbone.js to upskill himself. Based on his new skill set, a new project has been assigned to him.

The company has asked him to write a program to organize a JavaScript code using the MVC structure.

In this lesson, we will learn how to solve this real-world scenario to help Joe effectively complete his task.

Learning Objectives

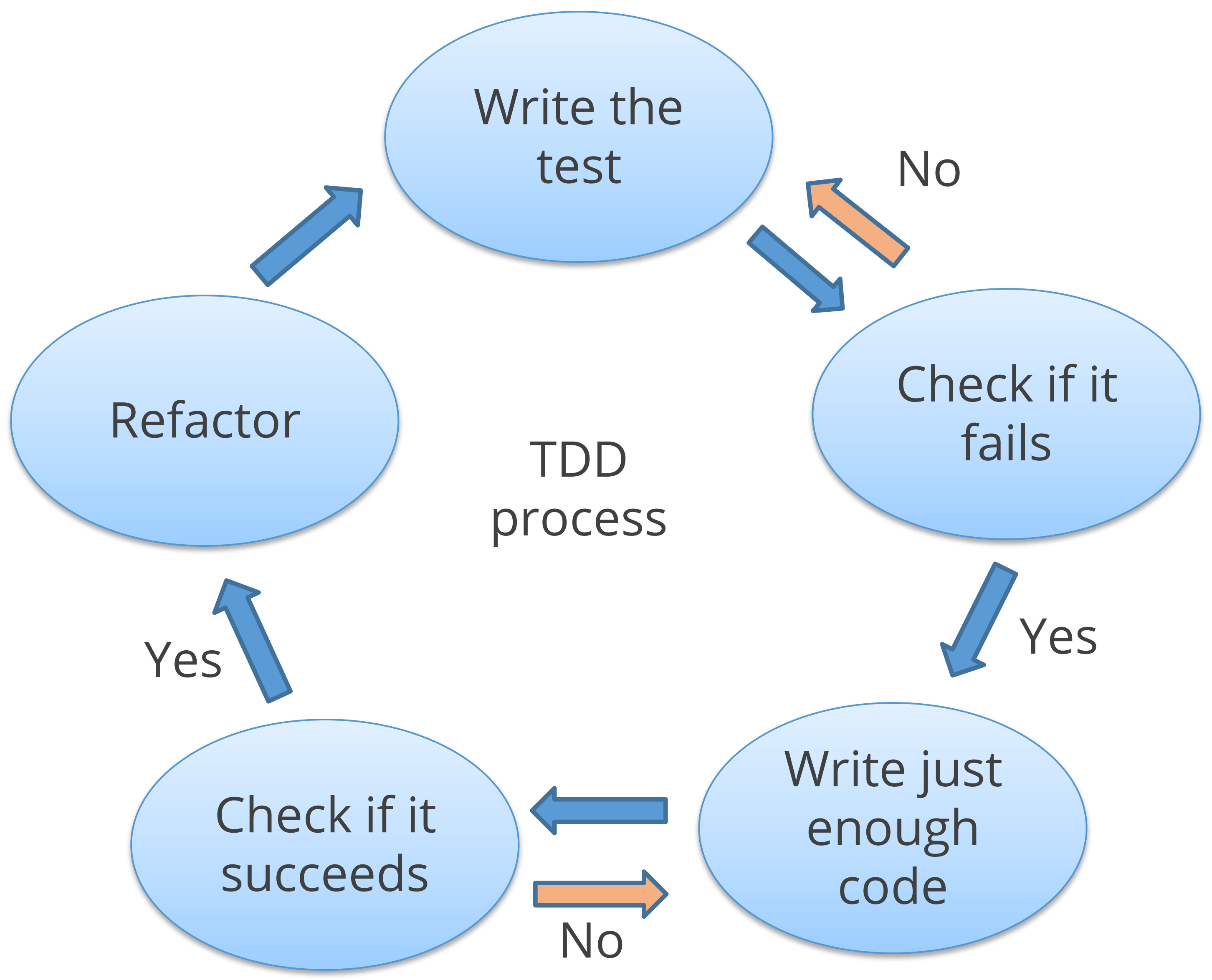
By the end of this lesson, you will be able to:

- List the differences between TDD and BDD
- Write simple test case with Jasmine
- Automate the process and run test with Karma
- Test the routers

Basics of Testing

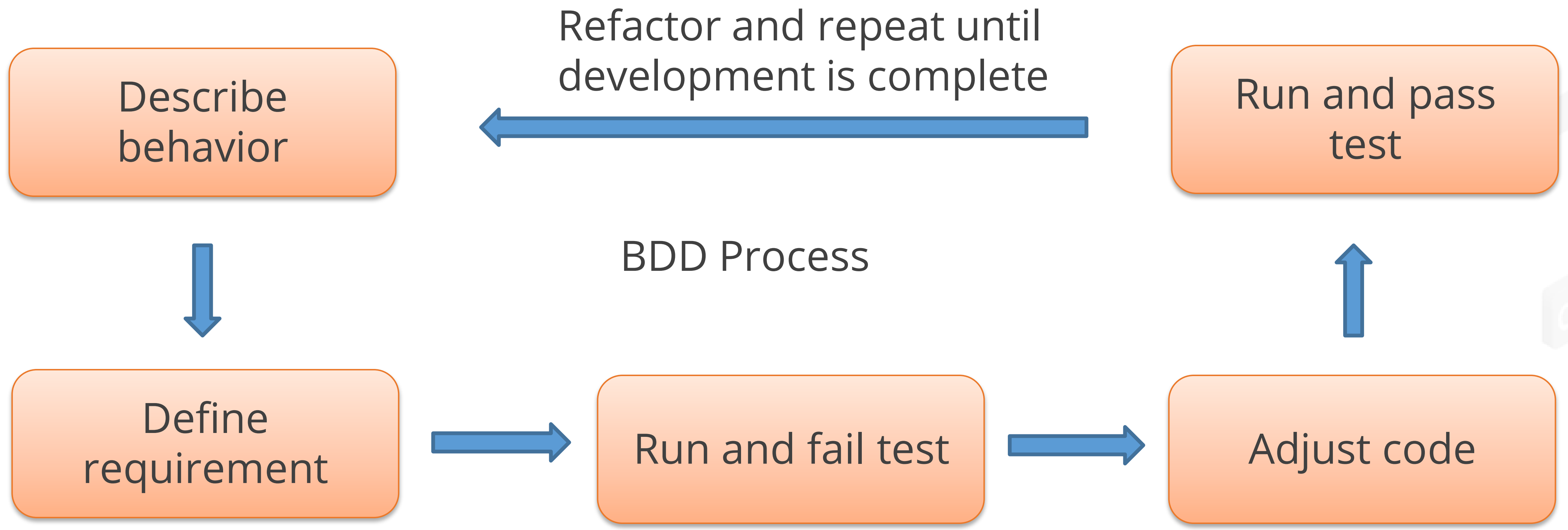
Test-Driven Development (TDD)

TDD is an agile, test-first programming model where a failing unit test is first written and then just enough code is written or modified to pass the test.



Behavior-Driven Development (BDD)

BDD is an agile software development technology that has originally evolved from TDD. It designs and develops an application around a behavior preferred by users.



TDD vs. BDD

TDD	BDD
TDD stands for Test-Driven Development	BDD stands for Behavior-Driven Development
TDD focuses on the implementation of the functionality	BDD focuses on the behavior of an application for the end user
TDD test cases are written in programming languages like Ruby, Java, etc.	BDD test cases are written in English like human-readable format by the stakeholders
Collaboration is required only between the developers	Collaboration is required between all the stakeholders
Tests in TDD can be understood by people who have programming knowledge	Tests in BDD can be understood by any person without any programming knowledge
Tools which support TDD: Junit, TestNG, and NUnit	Tools which support BDD: SpecFlow, Cucumber

Investigating Toolshed

Jasmine framework will used for testing along with the following components:

Test Extension -> jasmine-jQuery

- Extends the functionality of Jasmine
- Helps to manage the DOM
- Adds custom matchers to test properties of elements in DOM

Test Helper -> Sinon.js

- Standalone testing library
- Provides helpers, such as Stubs, Mocks, and Spies

Test Runner -> Karma

- Automated Test Runner
- Runs multiple tests on multiple browsers at the same time
- Makes the whole testing cycle easier

FULL STACK

Understanding Backbone.js

Backbone.js

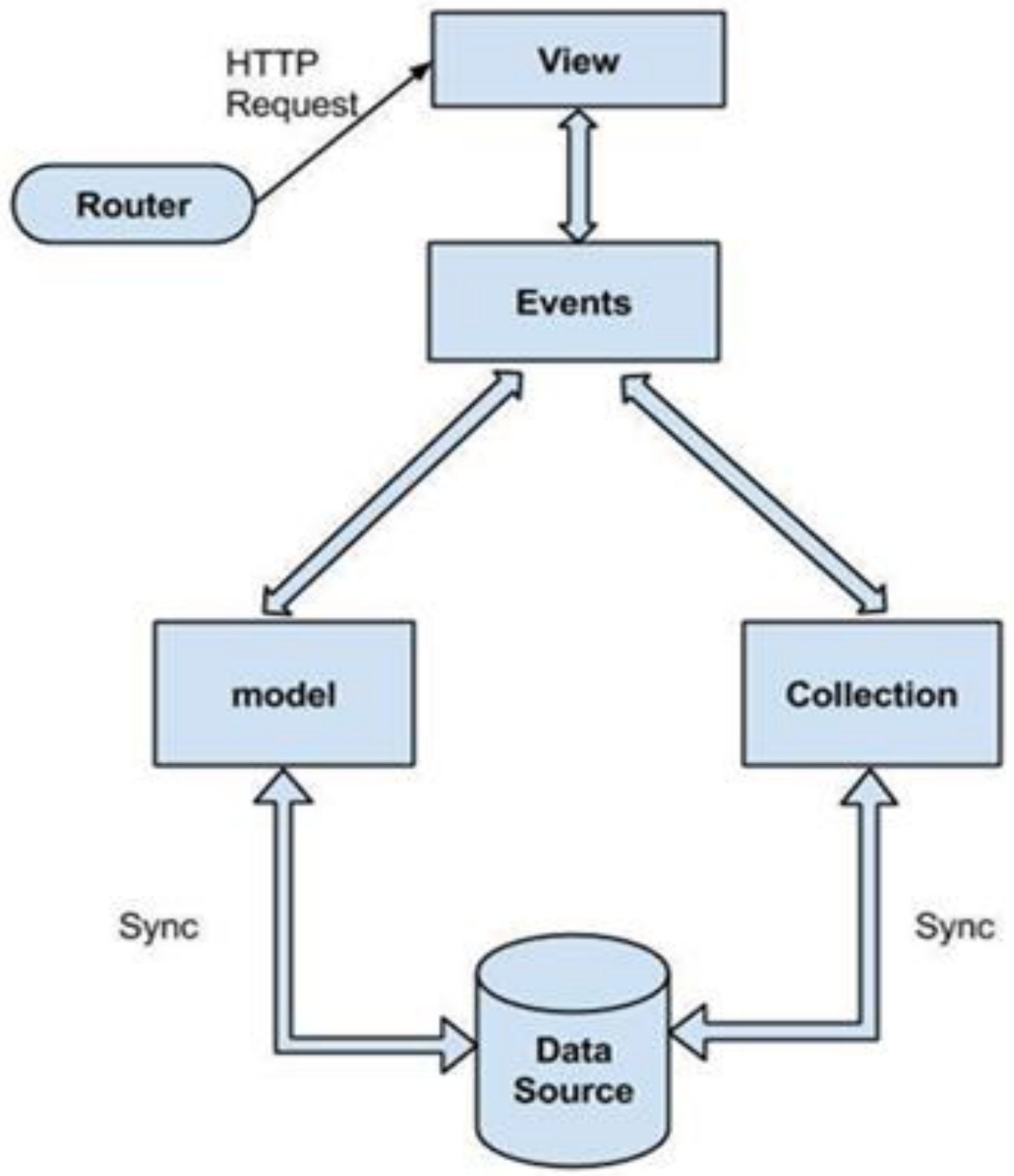
Backbone.js is a lightweight JavaScript library based on *Model-View-Presenter* design paradigm that gives structure to the client-side application that runs in a web browser.

It provides different types of building blocks like models, views, events, routers and collections for assembling client-side web applications.

It helps to organize the code and acts like a backbone for your project.

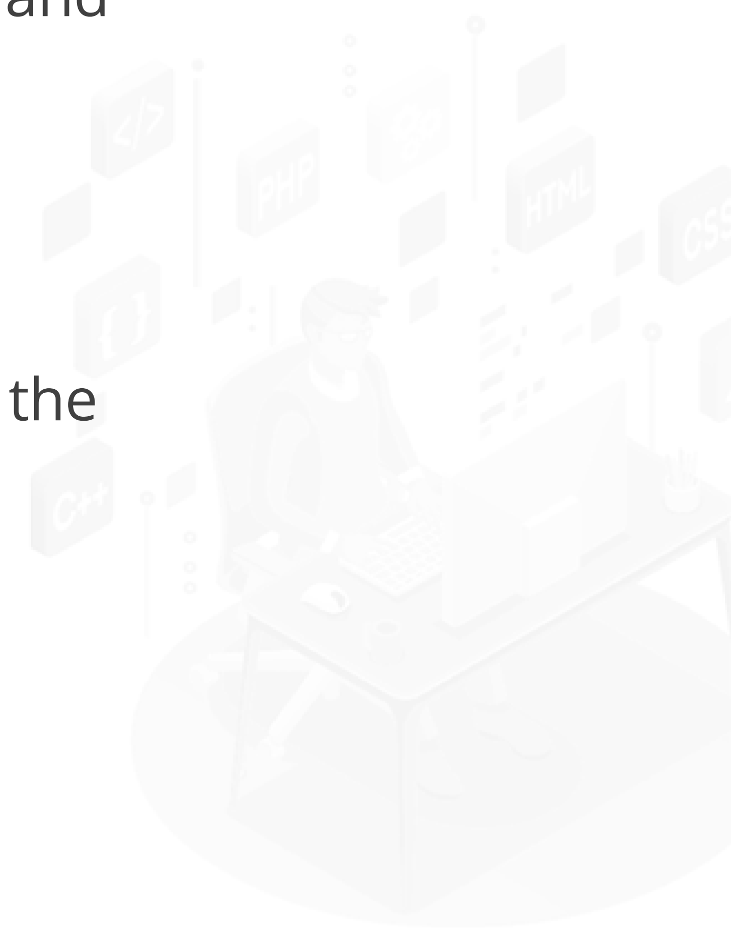
It is a free and open source library and contains over 100 available extensions.

Backbone.js Architecture



Benefits of Backbone.js

- Allows developers to develop one page applications and front-end much easier and better using the JavaScript functions.
- Gets tied easily with the back-end.
- Updates the HTML of your application automatically when the model changes.
- Manages the data model which includes the user data and displays that data at the server side with the same format written at client side.
- Has soft dependency with jQuery and hard dependency with Underscore.js.



FULL STACK

Overview of Jasmine

simplilearn

Introduction to Jasmine

- Jasmine is a Behavior-Driven Development (BDD) test framework.
- Allows you to write automated JavaScript unit tests.
- Does not depend on any other JavaScript framework.
- Does not require any DOM.
- Is an open-source framework and is easily available in different versions.

Writing Simple Test

Duration: 40 min.

Problem Statement:

You are given a project to write a simple test.

Assisted Practice: Guidelines

Steps to write simple test:

1. Download the copy of Jasmine
2. Write a test using Jasmine
3. Run test
4. Push the code to GitHub repositories

Testing Models

Testing Models

We are going to use “test-after” approach in this section. To test a model, we will:

Have folders and files

- There is a folder named “spec”.
- Inside this folder, we will have “models” and “views” folders that contain tests for models and views.

Use a testrunner

- We will use the Jasmine test framework.
- It is very easy to set up.
- We require jasmine.js and jasmine.css

Testing Models

Have Jasmine test skeleton

- Jasmine is a BDD test framework.
- Testing with Jasmine is easy.

Write test cases

- Test cases are written in this process.

Check test reports

- We are going to run the respective html file in the browser.

Testing Models

Duration: 20 min.

Problem Statement:

You are given a project to test a model using Backbone.js.

Assisted Practice: Guidelines

Steps to test models:

1. Write a test using Jasmine
2. Run our test
3. Push the code to GitHub repositories

Automating the Process

Overview of Karma

Karma is widely used to create productive testing environment for the developers.

It is highly configurable and can be integrated with various continuous integration packages.

It is a testrunner for JavaScript that runs on Node.js.

It is an open-source and easy debug environment.

It is suitable for testing AngularJS or any other JavaScript-based projects.

Automating the Process

Duration: 40 min.

Problem Statement:

You are given a project to install Karma, run Karma in browsers, and run a test with Karma.

Assisted Practice: Guidelines

Steps to automating tests using Karma:

1. Install Karma
2. Create karma.conf.js file
3. Break down the config file
4. Create a simple Karma test
5. Run the test in automatically launched browser
6. Open manually in any other browser
7. Push the code to the GitHub repositories

Depending on AJAX

Duration: 60 min.

Problem Statement:

You are given a project to handle AJAX request in tests.

Assisted Practice: Guidelines

Steps to handle AJAX requests in tests:

1. Add a collection
2. Write collection methods
3. Incorporate AJAX into code
4. Handle AJAX requests in tests
5. Push the code to the GitHub repositories

Testing Views

Testing Views

While testing a view, you should focus on:

Initialization

- Testing the view is provided with all required inputs like: model, collection, and localization texts.
- If view is not able to render test without some option, you should check if that exception is thrown.

Rendering

- Test if the required html has appeared in the view.
- Check for the presence of major DOM elements with the right styles.

Testing Views

Events	Model changes and persistence
• Test if the view is handling DOM events.	• Test if the changes in view are propagated in the model. If view is about to persist the model, then that could be tested as well.

Testing Views

Duration: 40 min.

Problem Statement:

You are given a project to test views.

Assisted Practice: Guidelines

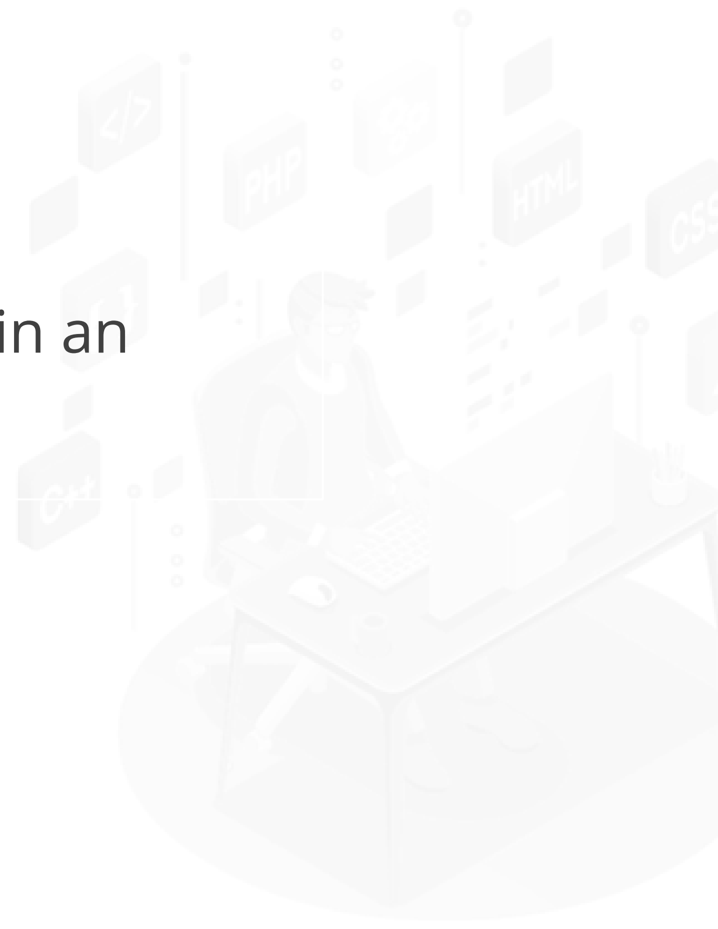
Steps to setup view and test:

1. Write basic view test
2. Setup basic view
3. Run test with DOM dependence and custom assertions
4. Prepare Karma to use the template within your tests
5. Install and include jasmine-jquery
6. Use jasmine-jquery in your test
7. Add handlebars to the view
8. Add notes to the code
9. Push the code to the GitHub repositories

Testing Routers

Router

- The backbone router is used to represent application state as a sharable and navigable URL.
- The user can get back to the same application using the shared URL.
- The router is responsible for orchestrating data fetching from backend, drawing complicated potential views from the parameters of the URL, and creating state in an application.



Testing Routers

Duration: 30 min.

Problem Statement:

You are given a project to test the routers.

Assisted Practice: Guidelines

Steps to demonstrate test for router:

1. Write a test for route
2. Write an implementation of the route setup
3. Push the code to the GitHub repositories

Key Takeaways

- TDD focuses on implementation of functionality, where BDD focuses on behavior of the application.
- Jasmine is an open source testing framework that is able to test any JavaScript application.
- Karma is an open source and easy to debug testing environment.

Backbone.js Testing for Movie Library

Problem Statement: **Duration: 40 min.**

You have been asked to create a backbone js application that can act as a consolidated library of movies with information like the name, ratings, genre, comments, etc. You are also required to automate the unit testing of this application with the help of Fixtures and testing tools for javascript like Jasmine.