CROSS | OVER

Congratulations on making it to this stage of the evaluation. You are obviously very talented as very few people make it to this stage. As we've stated earlier, the companies we represent receive 100s of resumes for any given role and it is through these difficult assignments where you can differentiate yourself and be noticed. After completion of this final 'real scenario' assignment - there will be a quick technical interview on your delivery then you are ready to be hired.

The project is scoped to be simple and reasonable in size to enable you to demonstrate your enterprise - class skills. Though this is a fictitious example, this scenario is very similar to what you may encounter in your job.

Technical Trial Objective

You are supposed to develop the frontend of a single page application. It is a video portal, where users can login, see video listings, navigate to a single video, rate videos and can perform all media related tasks e.g. Play, Pause, Adjust volume and seek video position. (You can use the default HTML5 video player controls)

The code for backend API and instructions for its deployment can be found at this zip file download.

Pre-Requisites

Client

You can choose any of the frameworks below to develop your client app:

- AngularJS (1.x or 2.x)
- EmberJS
- REACT
- Backbone
- Grunt/Gulp
- Karma/Jasmine

Server

- NodeJS and npm
  - express
  - body-parser
  - morgan
  - mongoose
- MongoDB

Functional Requirements

1. Develop a single page application by using one of the allowed MVV* frameworks.

2. Design the UI, which should be motivated by the provided visuals.

3. Implement user authentication. The content of this application should not be visible to public.

4. User should be able to see video listings on index page. Only first 10 videos should be loaded initially.

5. Lazy loading should be implemented i.e. More videos should appear as the user scrolls down the listing.

6. Users should not be able to play more than 1 video simultaneously. Playing a video should pause all other videos.

7. Users should be able to rate videos, an overall rating for each video should also be displayed.

8. Users should be able to open the video details page by clicking on video title.

9. REST API should be consumed.

10. Unit tests with at least 50% code coverage should be provided.

Non-Functional Requirements

1. Code should be well documented by comments.

2. Exception handling should be done, where necessary.

3. Code should be well organized into files and folders.

4. UI design should be clean and polished.

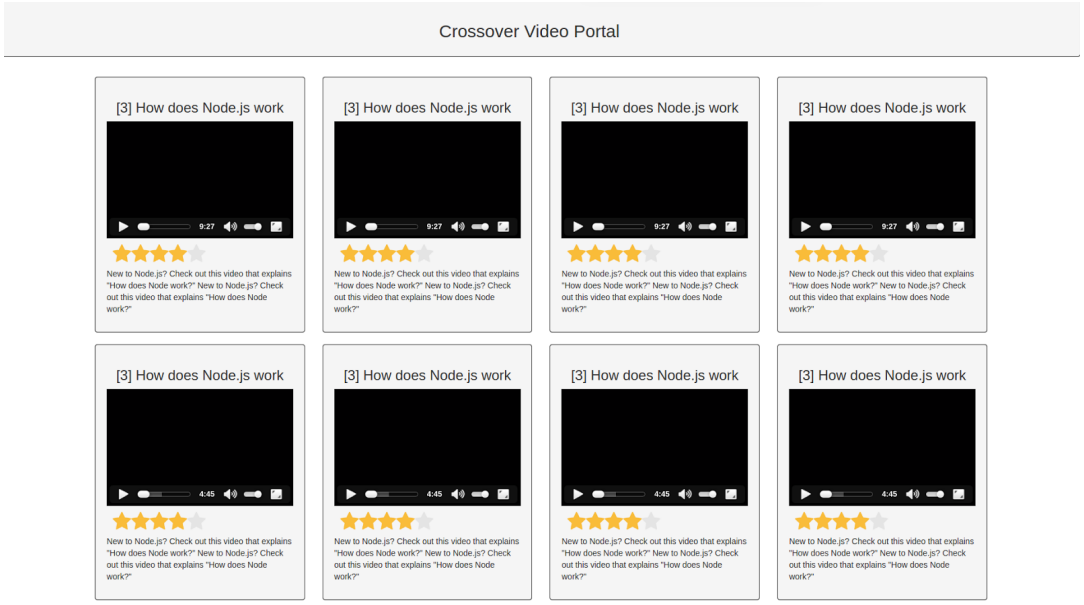
5. CSS animations should be used to make the application more appealing.

6. UI should be cross-browser compatible.

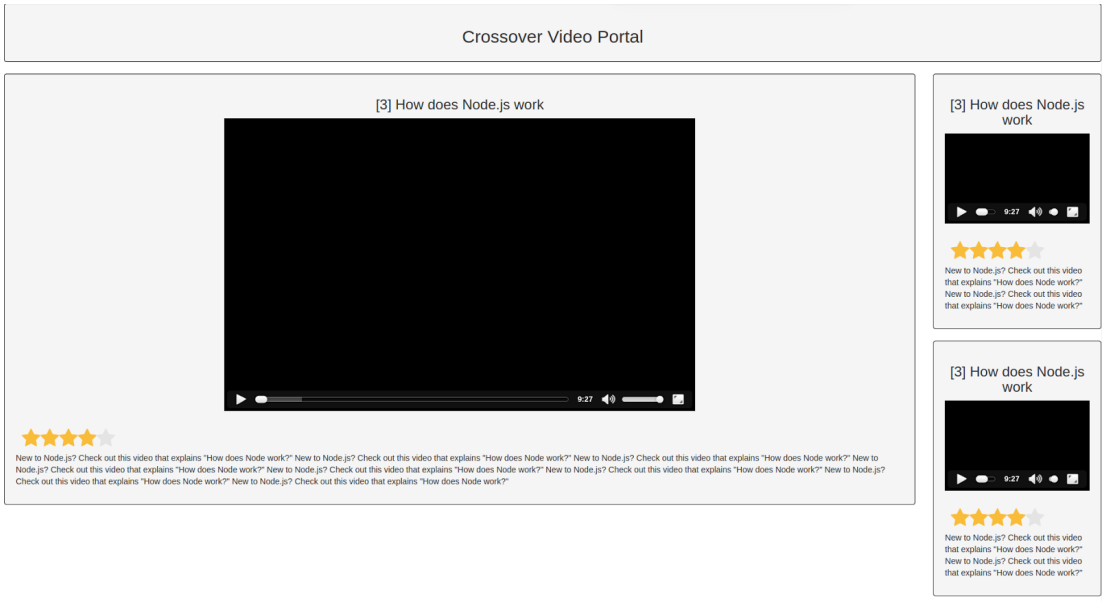
7. UI should be cross-device compatible.

Visuals

The provided visuals are just for reference. Your design should be inspired by these visuals but feel free to exhibit your creative designing skills.



Index page, with video listings



Video detail page

Backend:

Download the backend code from [this link.]

MongoDb should be running.

Update the DB details in config.js

Run ‘npm install’ to download npm packages.

Run ‘npm start’ to start the backend API.

By Default the backend will start web server on http://localhost:3000/

Backend is designed to serve your client code. Simply copy your code inside the client folder, and your app would be accessible at root ’/ ‘ (with default configuration the URL will be http://localhost:3000/)

Authenticate:

In Initial setup only three users are created - “ali”, “tom” and “harry” with password “password” for all.

This API call is used to authenticate the user to enter the application.

JSON object with with following attribute needs to be sent in Request:

- username.
- MD5 encrypted password.

JSON object with Status, Session ID and username is returned in Response.

URL: http://localhost:3000/user/auth

Method: POST

Examples:

Input:

```
{"username":"ali", "password":"5f4dcc3b5aa765d61d8327deb882cf99"}
```

Response:

```
{"status":"success","sessionId":"a8t9Rr9bjWD2InfeFLbNS3FNg5mnFqiV","username":"ali"}
```

sessionId is to be sent with each request after login.

Logout:

This API call is used to logout the user from application.

Query string should contain the following parameters:

- sessionId

JSON object with Status is returned in Response.

URL: http://localhost:3000/user/logout

Method: GET

Examples:

Request URL:

```
http://localhost:3000/user/logout?sessionId=CLr7NWvDvdyy1h9Uhtce0CaO4lL09d0z
```

Response:

```
{"status":"success"}
```

Get Video Listings:

This API call is used to get the video listings.

Query string should contain the following parameters:

- sessionId
- skip [optional] (number of records to skip from start)
- limit [optional] (number of records to fetch)

JSON object with status and videos data is returned in Response.

URL: http://localhost:3000/videos

Method: GET

Examples:

Request URL:

http://localhost:3000/videos?sessionId=qOhmSmXoLuFv6g1YvvXOU4UJ2nuI11jA&skip=1&limit=1

Response:

{"status":"success","data":[{"_id":"5757e6e41b0a244b256ac1d5","name":"[1] Google Cardboa

Get Single Video:

This API call is used to get details of a single video.

Query string should contain the following parameters:

- sessionId
- videoId

JSON object with status and video data is returned in Response.

URL: http://localhost:3000/video

Method: GET

Examples:

Request URL:

http://localhost:3000/video?sessionId=CqqAfowuZLzXHyViPaYzzyOU3dGPCFaG&videoId=5757e6e41

Response:

{"status":"success","data":{"_id":"5757e6e41b0a244b256ac1d5","name":"[1] Google Cardboar

Rate Video:

This API call is used to add user rating to a single video.

Query string should contain the following parameters:

- sessionId

JSON object with with following attribute needs to be sent in Request:

- videoid.
- rating. (ranges from 1 to 5)

JSON object with status and video data is returned in Response.

URL: http://localhost:3000/video/ratings

Method: POST

Examples:

Request URL:

http://localhost:3000/video/ratings?sessionId=CqqAfowuZLzXHyViPaYzzyOU3dGPCFaG

Input:

```
{"videoId":"5757e6e41b0a244b256ac1d7","rating":"5"}
```

Response:

```
{"status":"success","data":{"_id":"5757e6e41b0a244b256ac1d7","name":"[3] How does Node.js
```

To be evaluated :

- The quality of your output.
    - Code Modularisation
    - Code quality
    - Documentations
    - Best Practices of Technology Applied (MV* JS framework mandatory)
- User Interface of App
    - Easy to use
    - Attractive
    - UI responsiveness from Mobile to tablet to desktop.
    - Cross-Browser Compatible (IE+10 is sufficient)

    - Creativity in applying interactivity particularly in animation
- Requirement Fulfillment.
- Recorded Demo (mandatory)

Delivery / What to submit

Please, read and follow this section carefully. Any delivery that does not follow this section will score much less or simply won't be evaluated. Verify all the files and folders before submission, any missing file or folder might result in less scores.

Delivery for this assignment should consist of:

Archive named <your_name-Javascript Architect>.zip containing the following:

- README.txt -> containing report about work done and how to run your application.
- Wink Folder -> Wink video(s) shows you application running, and perform required functionality (You can use any recording application if you couldn't use wink).
- Code Folder -> contains your Code & unit tests.

Check that the size of the archive is less than 250MB. If not, reduce the size of the demo video by removing similar frames and remove the dependencies.

ATTENTION! YOUR APPLICATION WILL BE REJECTED IF IT:

- Does not compile

- Does not contain unit tests

- Unit tests are failing