# ExamQuestions2

## Why would you consider a Scripting Language as JavaScript as your Backend Platform.

- Frontend and Backend would most likely know the language.

- Reduces number of languages used overall since u will most likely use something javascript related in frontend.

- Javascript can do callbacks which makes your code asynchronous

# Explain Pros & Cons in using Node.js + Express to implement your Backend compared to a strategy using for example Java/JAX-RS/Tomcat

## Pros:

- code sharing between frontend and backend.

- growing number of good libraries such as npm

- You don't actually have to write in Javascript - many languages compile to Javascript these days

## Cons:

- documentations are often very sparse

- libraries, while many, still are fewer comparing against others

# Node.js uses a Single Threaded Non-blocking strategy to handle asynchronous task. Explain strategies to implement a Node.js based server architecture that still could take advantage of a multi-core Server.

- Since v0.6.X Node.js has included the cluster module straight out of the box, which makes it easy to set up multiple node workers that can listen on a single port. Note that this is NOT the same as the older learnboost "cluster" module available through npm.

https://nodejs.org/docs/latest/api/cluster.html

- Method 1:

round-robin approach, where the master process listens on a port, accepts new connections and distributes them across the workers in a round-robin fashion, with some built-in smarts to avoid overloading a worker process.

- Method 2:

the master process creates the listen socket and sends it to interested workers. The workers then accept incoming connections directly.

# Explain, using relevant examples, the Express concept: middleware.

Middleware is any number of functions that are invoked by the Express.js routing layer before your final request handler is, and thus sits in the middle between a raw request and the final intended route. We often refer to these functions as the middleware stack since they are always invoked in the order they are added.

app.use(function(req, res, next) {

  console.log('%s %s', req.method, req.url);

  next();

});

app.get('/', function(req, res, next) {

  res.send('Hello World!');

});

app.get('/help', function(req, res, next) {

  res.send('Nope.. nothing to see here');

});

Here we have added a new function to invoke with every request via app.use(). There are some important things to notice about this new function:

* Middleware is a function, just like route handlers, and it is invoked in much the same way.
* The function signature looks *identical* to the one we used in our routes.
* We added this function *before* our two route handlers because we want it to execute before either one.
* We have full access to the same request and response objects that will find their way to the routes.
* We used a third parameter, called next as a function to indicate that our middleware was finished.
* We can add more middleware above or below using the same API.