**Introduction**

Node.js runs on the V8 JavaScript engine, which enables it to launch programs written in Java independently from the browser, while at the same time allowing it to work with file systems, OS, and devices. In particular, Node.js can be used to create various network applications.

Here’s a basic example of Node.js-based code:

var http = require('http');

var requestHandler = function(request, response) {

response.writeHead(200, {'Content-Type': 'tex/plain'});

response.end('Hello world');

};

var server = http.createServer(requestHandler);

server.listen(8080);

This code creates an HTTP server that listens on 8080 port.

The main Node.js feature is the use of a non-blocking input/output event model. At its core lies an “event cycle”, a non-stop process that examines defined sources for the presence of new events. If any new events are found, Node.js launches the handler that was previously signed to the specific event. Thus, every operation occurring in the application is processed asynchronously depending on arising events. Since Node.js is a single-threaded application, every asynchronous event is executed one after the other and not simultaneously.

Events.EventEmitter is responsible for working with events, and can be obtained by calling require('events').EventEmitter. In the given example, the source of events is a server object. With every query, the server object creates the request event, which is processed by the function called requestHandler.

The other Node.js feature is that many base objects are streams. Since Node.js can work with large data arrays or data that becomes available over time, streams are suitable for working with these tasks. Node.js has different kinds of streams: readable, writable, duplex (can read and write simultaneously) and transforms (a special kind of a duplex stream). Streams feature a large set of useful functions, such as pause, resume, and pipe.

Since Node.js at its core is a JS-engine wrapper, many functions needed for the external environment to interact with Node.js are implemented in the form of plugins. The standard Node.js pack has a number of frequently used plugins, like OS plugins, File System (FS) plugins, Net plugins etc. The given example uses the HTTP plugin, which helps with creating an HTTP server or HTTP client. The module connects to the application through the “require” line. There are lots of third-party modules for Node.js, they can be connected and configured by using the Node.js packet manager.

Example of package installation:

npm install async

In this particular case, the async module is being installed locally. The annotation in the form of the package.json (can be found in the root directory of the project) file and is used for project and dependencies description. Here’s an example:

{

'name': 'Test

'version': '0.0.1',

'description' : 'My test project',

'dependencies': {

'async' : '\*',

'express' : '3.2'

}

}

The good thing about writing web applications is that you can use libraries and codes on the client and server sides alike. For example, you can use templates and libraries common to both parties for data transformation and conversion.

**Task:**

Implement a web application called “chat”, which has the following functions:

* Registering and authorizing users
* Creating chat rooms
* Transferring messages in real time
* Saving and displaying message history
* File transfer

A “templates” folder with HTML pages is attached to the task.

You can use them as template sources.

Templates

The main factor is not the application’s efficiency, but its structure and the adequacy of using different components in order to perform specific tasks.

**Recommended Technologies**

We recommend the following libraries, yet it’s possible to use others:

* Express: basic framework for creating applications;
* Async: module for working with the asynchronous aspects of JavaScript code;
* Socket.IO: library for providing a real-time connection to the server;
* Mongodb, Mongoose: module for connecting and an object wrapper for Mongodb database;
* Winston: logging module;
* Jade, Swig, EJS: template modules.

**Materials:**

<http://nodejs.org/api/> - Node.js documentation

<http://www.joyent.com/developers/node> - Must-read list of articles concerning the main components of Node.js

<http://nodeguide.ru/doc/> - has some useful Node.js articles

<http://expressjs.com/> - Express framework page with documentation

<http://socket.io/> - Socketio library

<http://npmjs.org/> - Node.js module pack

Optional:

<http://codeschool.com/courses/real-time-web-with-nodejs> - Short yet informative Node.js video course