Compile server

Project to serve back-end of Blockly4Arduino.
Compile your sketches or build your project and return .hex-file of requested code.
This project uses the arduino-builder tool used by the Arduino IDE.

Requirements

- Linux or Ubuntu OS (Host or VM);
- Node.js installed (https://nodejs.org/en/download/);
- npm installed (if not installed with Node.js);
- Latest version of the Arduino IDE installed (https://www.arduino.cc/en/Main/Software).

Setting up the server

- Download or clone following GitHub-directory: https://github.com/RMeurisse/ArduinoBuilder;
- `cd` to downloaded directory;
- Run `(sudo) npm install` in the current directory. This will install all dependencies specified in the manifest.json-file;
- Change the file 'config.js' to specify your local paths and variables:
    - Local './arduino-directory': this directory will contain the extra libraries you downloaded with the Arduino IDE;
    - Local './arduino-1.8.5-directory': this directory contains all the files used by the arduino-builder tool;
    - Local 'temporary'-directory: this is the directory where the arduino-builder tool will store it's temporary files, this can be a directory you created or you can use the '/tmp'-directory of Linux/Ubuntu.
- Run `(sudo) node server.js`;
- Go to: `http://localhost:{Port}/`. With {Port} being the port specified in the 'config.js'-file;
- Write your code in the textbox and verify your code or upload it directly to a connected Arduino board;
- (To automatically start the server at start-up, see next section).

Automatic start server at start-up of the system

- In the downloaded directory, find the file 'blocklyserver.service';
- Edit the following line in the 'blocklyserver.service'-file to refer to the location of the server.js-file:

```
ExecStart=[PATH TO SERVER.JS FILE]
```

- This service-file will execute the server at startup or restart the server automatically if it crashes;
- Before continuing: make sure the first line of the server.js-file contains the following: `#!/usr/bin/env node`. This command will make sure the system knows that file needs to be executed with node. Also make sure the server.js-file is executable by doing the following: `(sudo) chmod +x [PATH TO SERVER.JS FILE]`;

- Copy or move this file to the startup-directory. For linux: `(sudo) cp [PATH TO SERVICE FILE] /etc/systemd/system/`;
- Enable the service: `systemctl enable blocklyserver.service`;
- Start the service: `systemctl start blocklyserver.service`;
- Verify it's running: `systemctl status blocklyserver.service`;
- If there are problems, you can check the errors in the log: `less /var/log/syslog` or messages with `journalctl -u blocklyserver.service`.

Optional: The above steps will run the service as root. If you want to run the server from a specific user only, follow the steps below;

- Change the following lines to specify the user:

  ```
  User=[USERNAME]
  Group=[YOUR GROUP]
  ```

- After making changes to the service file, reload it: `systemctl daemon-reload`;
- Restart the service: `systemctl restart blocklyserver.service`;
- Check status with: `systemctl status blocklyserver.service`.

Development

If you want to integrate the given index.html page into your own website.

POST request to server

Your website should send a HTTP POST-request with JSON-data of the following format:

```
{
    'data': 'void setup(){} void loop(){}',
    'boardName': 'arduino:avr:uno'
}
```

The 'data'-property should contain your program (text written in the textbox) and the 'boardName'-property should contain
the name of the board you want to compile code for (see 'Accepted board-types' below for the name).

Response from the server

When sumbitting a request, the server will return a response message containing JSON-data of the following format:

```
{
    'hex': '...',
    'out': '...',
    'err': '...'
}
```

The 'hex'-property contains the compiled program in hex-format, ready to be uploaded onto the board.
The 'out'-property contains all the console messages of the compiler while the 'err'-property contains the
error messages if there was an error, otherwise this will be empty.

Accepted board-types/names

The following table displays the board string parameters for the POST-request to the server. The last column is the name for the board that the extension requires. The board string is different for the builder from the extension.
Only the standard board strings are given for the arduino-builder. If you want to add other boards, use the Arduino IDE to include them. When you verify a program, the console outputs the commands used, search for the option `-fqbn [arduino:avr:...]`. This option is the one you will have to include in your website.

Programmer	**Arduino-builder Board String**	**Extension Board String**
Arduino Uno	`arduino:avr:uno`	`uno`
Arduino Mega 1280	`arduino:avr:mega:cpu=atmega1280`	`mega`
Arduino Mega 2560	`arduino:avr:mega:cpu=atmega2560`	`mega`
Arduino ADK	`arduino:avr:megaAdk`	`adk`
Arduino Leonardo	`arduino:avr:leonardo`	`leonardo`
Arduino Micro	`arduino:avr:micro`	`micro`
Arduino Nano 328	`arduino:avr:nano:cpu=atmega328`	`nano`
Arduino Lilypad USB	`arduino:avr:lilypad-usb`	`lilypad-usb`
Arduino Duemilanove	`arduino:avr:diecimila:cpu=atmega168`	`duemilanove168`
Arduino Yun	`arduino:avr:yun`	`yun`
Arduino Esplora	`arduino:avr:esplora`	`esplora`
RedBearLab Blend Micro	``	`blend-micro`
Tiny Circuits Tinyduino	``	`tinyduino`
SparkFun Pro Micro	`arduino:avr:pro:cpu=16MHzatmega328`	`sf-pro-micro`
Qtechknow Qduino	``	`qduino`
Pinoccio Scout	``	`pinoccio`
Femtoduino IMUduino	``	`imuduino`
Adafruit Feather 32u4 Basic Proto	``	`feather`
Arduboy	``	`arduboy`
Adafruit Circuit Playground	`arduino:avr:circuitplay32u4cat`	`circuit-playground-classic`

Functions used by the webpage

The webpage uses several functions to compile the given code and flash the hardware through the extension.

```
/**
 * Function checkExtension will poll if the extension is installed or not.
 * The function uses short-lived connections with the extension to do this
(chrome.runtime.sendMessage()).
 * Steps:
 *  1. Send message to extension with option 'check';
 *  2. Extension will respond if installed, otherwise
chrome.runtime.sendMessage() will return false.
 */
function checkExtension()
```

```
/**
 * Function populatePorts will add connected devices to the select box.
 * Steps:
 *  1. Send message to extension with option 'ports';
 *  2. Extension will retrieve all ports that have hardware connected;
 *  3. Extension sends message back containing JSON-data with the list of
ports;
 *  4. This function will populate the drop-down list with the returned ports.
 */
function populatePorts()
```

```
/**
 * Function handleSubmit() will load a hex-file from the local PC
 * and post it to the extension to flash the device.
 * Steps:
 *  1. Open filesystem so the user can select a .hex-file;
 *  2. Make JSON-Ojbect containing the contents of the selected file, the
Arduino board name and the selected port;
 *  3. Send the object to the extension with a port-object (port.postMessage()
is a long-lived connection because it can take some time);
 *  4. Extension will respond if the flashing was succesful.
 */
function handleSubmit()
```

```
/**
 * Function verifytxt() will return whether or not the code is compilable.
 * This function will NOT flash the device.
 * Steps:
 *  1. Makes a HTTP POST-request to the specified compiler with the code from
the textbox;
 *  2. Compiler sends a message back with success or error;
 *  3. Display the returned error.
 */
function verifytxt()
```

```
/**
 * Function uploadtxt() will flash the code in the textbox to the connected
device
 * Steps:
 *  1. HTTP POST-request to compile the code and to receive the hex-file;
 *  2. Send received hex-file to the extension;
 *  3. Extension will flash a connected device;
 *  4. Extension will return success if flashing was succesful.
 */
function uploadtxt()
```