# **FROM 24th July TO 30TH July**

# **Project ID:**

# **2021J\_BV01\_BCI Browser**

# **Project Title:**

# **Design and development of Brain Computer Interface Browser on Web and Mobile**

# **Summary:**

P300 spellers are among the most popular types of brain–computer interfaces (BCIs) and are extremely useful assistive devices that enable severely disabled patients to communicate. However, P300 speller performances should be further improved to translate laboratory designs into practical applications. We aimed to design a new speller paradigm that could evoke higher event-related potentials (ERPs) than traditional P300 spellers, thus improving the performance of BCI systems.

# **Detail:**

# **HOW TO STORE AND MANAGE LARGE DATABASE**

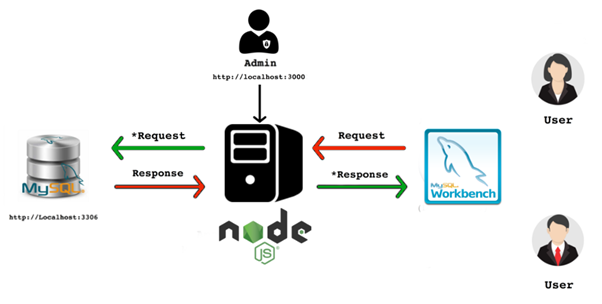
One of the biggest concerns when dealing with and managing databases is its data and size complexity. Often, organizations get concerned about how to deal with growth and manage growth impact because the database management fails off. Complexity comes with concerns that were not addressed initially and were not seen, or could be overlooked because the technology being currently used shall be able to handle by itself. Managing a complex and large database has to be planned accordingly especially when the type of data you are managing or handling is expected to grow massively either anticipated or in an unpredictable manner.

The size of the database matters as it has an impact on performance and its management methodology. How the data is processed and stored will contribute to how the database will be managed, whic.h applies to both in transit and at rest data. For many large organisations, data is gold, and growth in data could have a drastic change in the process. Therefore, it’s vital to have prior plans to handle growing data in a database.

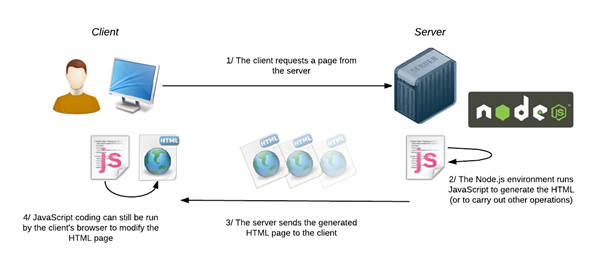
Normalizing tables maintains data integrity, reduces redundancy, and makes it easy to organize the data into a more efficient way to manage, analyze, and extract. Working with normalized tables yields efficiency, especially when analyzing the data flow and retrieving data either by SQL statements. Although concerns with normalized tables possess performance penalty and can slow the queries due to series of joins when retrieving the data. Whereas denormalized tables, all you have to consider for optimization relies on the index or the primary key to store data into the buffer for quicker retrieval than performing multiple disks seeks. Denormalized tables require no joins, but it sacrifices data integrity, and database size tends to get bigger and bigger.

Hash tables cannot do range retrieval, whereas B-Trees is very efficient for doing these types of searches and also it can handle large amounts of data. Dealing with 1000's or thousands of databases, using the right engine in combination of your queries and data that you want to retrieve and store shall deliver good performance. Given that you have predetermined and analyzed your requirements for its purpose for the right database environment

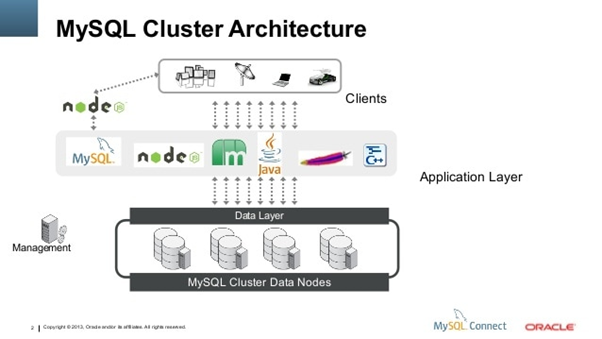
**NODEJS+MySQL**

****

**Figure: Nodejs+MySQL**

****

**Figure: Working process of Nodejs**

****

**SAMPLE CODE FOR DATABASE CONNECTIVITY USING NODEJS, EXPRESS**

const express = require('express')

const bodyParser = require('body-parser')

const mysql = require('mysql')

const app = express()

const port = process.env.PORT||5000;

// parse application/x-www-form-urlencoded

app.use(bodyParser.urlencoded({ extended: false }))

// parse application/json

app.use(bodyParser.json())

// MySQL

const pool = mysql.createPool({

host : 'localhost',

user : 'root',

password : '',

database : 'demo'

})

app.get('', (req, res) => {

pool.getConnection((err, connection) => {

if(err) throw err

console.log('connected as id ' + connection.threadId)

connection.query('SELECT \* from contactdetails', (err, rows) => {

connection.release() // return the connection to pool

if (!err) {

res.send(rows)

} else {

console.log(err)

}

// if(err) throw err

console.log('The data from contactdetails table are: \n', rows)

})

})

})

app.get('/:id', (req, res) => {

pool.getConnection((err, connection) => {

if(err) throw err

connection.query('SELECT \* FROM contactdetails WHERE id = ?', [req.params.id], (err, rows) => {

connection.release() // return the connection to pool

if (!err) {

res.send(rows)

} else {

console.log(err)

}

console.log('The data from contactdetails table are: \n', rows)

})

})

});

app.delete('/:id', (req, res) => {

pool.getConnection((err, connection) => {

if(err) throw err

connection.query('DELETE FROM contactdetails WHERE id = ?', [req.params.id], (err, rows) => {

connection.release() // return the connection to pool

if (!err) {

res.send(`Person with the record ID ${[req.params.id]} has been removed.`)

} else {

console.log(err)

}

console.log('The data from contactdetails table are: \n', rows)

})

})

});

app.post('', (req, res) => {

pool.getConnection((err, connection) => {

if(err) throw err

const params = req.body

connection.query('INSERT INTO contactdetails SET ?', params, (err, rows) => {

connection.release() // return the connection to pool

if (!err) {

res.send(`person with the record ID has been added.`)

} else {

console.log(err)

}

console.log('The data from contactdetails table are:11 \n', rows) })

})

});

app.put('', (req, res) => {pool.getConnection((err, connection) => {

if(err) throw err console.log(`connected as id ${connection.threadId}`)

const { id, name, tagline, description, image } = req.body

connection.query('UPDATE contactdetails SET name = ?, tagline = ?, description = ?, image = ? WHERE id = ?', [name, tagline, description, image, id] , (err, rows) => {

connection.release() // return the connection to pool

if(!err) {

res.send(`person with the name: ${name} has been added.`)

} else {

console.log(err)

}

})

console.log(req.body)

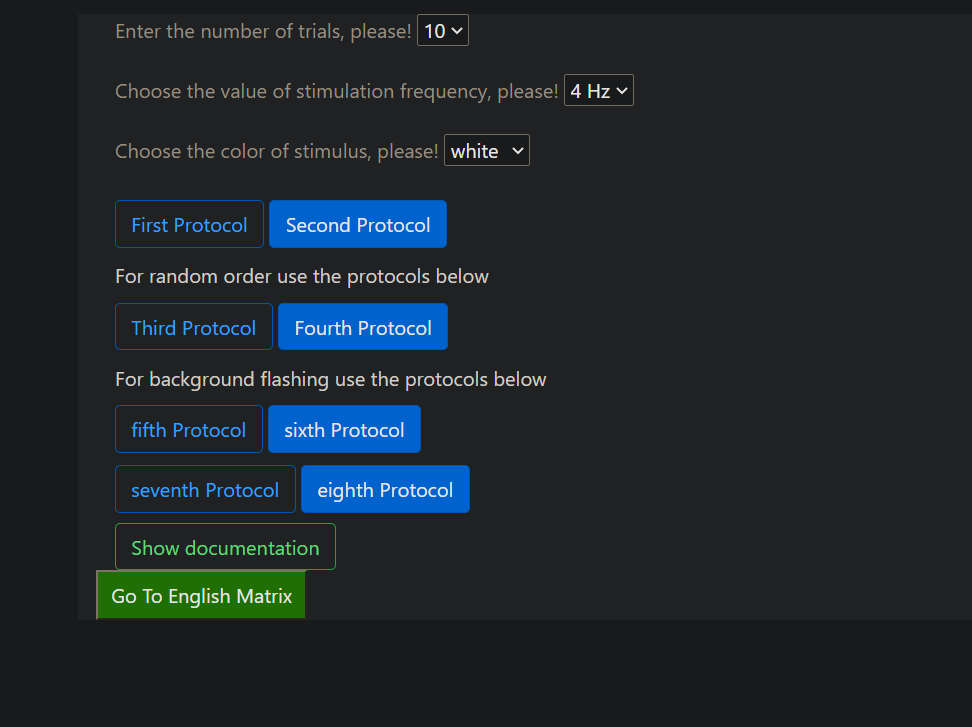
})

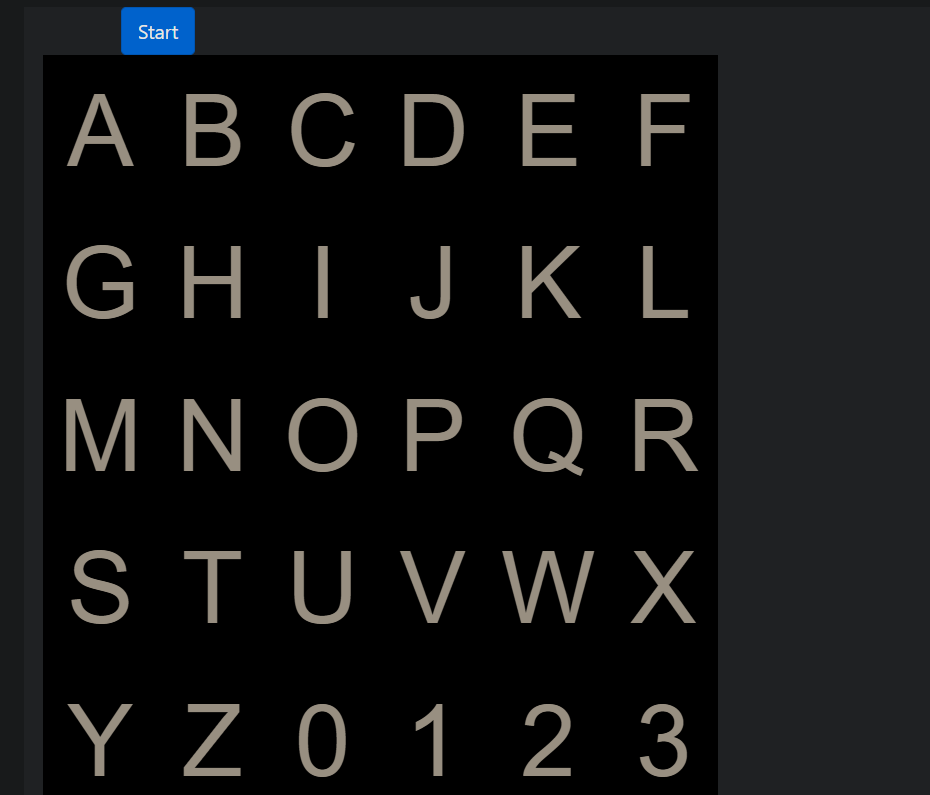
})

app.listen(port, () => console.log(`Listening on port ${port}`))

**});**

**UPDATES ON P300 SPELLER PROJECT**

****

****

The matrix and the buttons page was separated. Data selected from the buttons and dropdown lists was synced with the other page using Sessions object.

First page:

sessionStorage.setItem('number\_of\_trials', $('#number\_of\_trials').val());

console.log($('#duration\_of\_stimulus').val());

sessionStorage.setItem('duration\_of\_stimulus',$('#duration\_of\_stimulus').val());

sessionStorage.setItem('s-color', $('#s-color').val());

sessionStorage.setItem('firstProtocol', firstProtocol);

Second page:

if(sessionStorage.getItem("firstProtocol") == "true"){

document.getElementById("start1").style.visibility = "visible";

**CONCLUSION**

Work in progress