# **FROM 8th August TO 13th August**

# **Project ID:**

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

# **Project Title:**

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

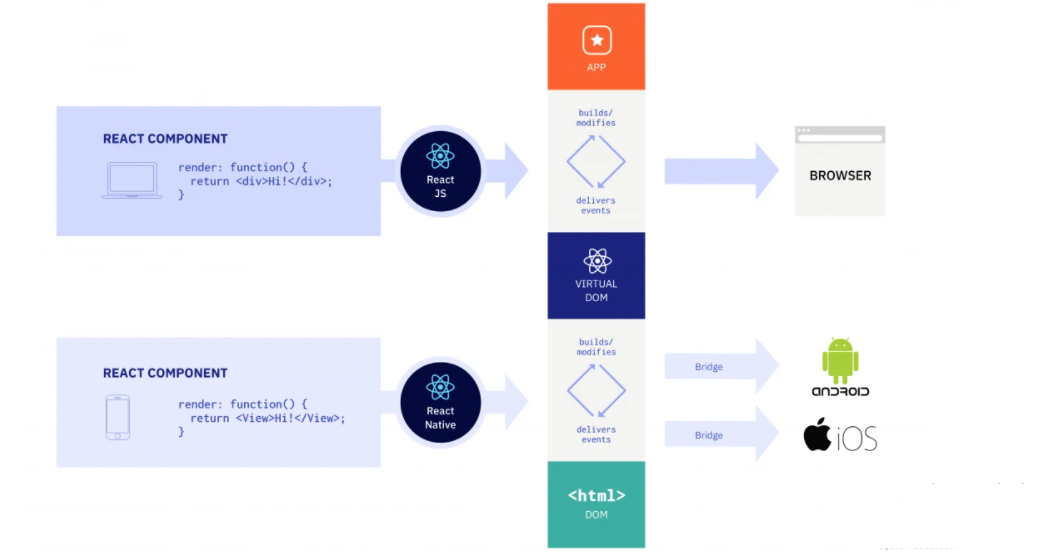
# **Summary:**

* Designing Front-End part of BCI browser(home page) using Reactjs.
* Updates on P300 SPELLER
* Visualization of data
* Knowledge Repository using POSTGRESQL

# **Detail:**

**REACT JS (Front-end)**

React. js is an open-source JavaScript library that is used for **building user interfaces specifically for single-page applications**. It's used for handling the view layer for web and mobile apps. React also allows us to create reusable UI components.



**REACTJS code for HOME PAGE**

import React from "react";

import { Button } from "reactstrap";

import "../App.css";

const Hero = () => (

<div>

<main className="cover-page" id="hero">

<section className="wrapped-page">

<div className="item-center">

<h1>WELCOME to <br></br>Brain-Computer Interface Browser</h1>

<h3>Repository | P300 SPELLER | Guide</h3>

<Button outline color="warning" href="#about">

Explore More

</Button>

</div>

</section>

</main>

</div>

);

export default Hero;

//**REACTJS code for NavBar**

import React from "react";

import {

Collapse,

Navbar,

NavbarToggler,

NavbarBrand,

Nav,

NavItem,

NavLink,

Container,

Button

} from "reactstrap";

import "../App.css";

class NavbarMain extends React.Component {

constructor(props) {

super(props);

this.toggle = this.toggle.bind(this);

this.state = {

isOpen: false

};

}

toggle() {

this.setState({

isOpen: !this.state.isOpen

});

}

render() {

return (

<div>

<Navbar

color="faded"

dark

expand="md"

fixed={`top`}

className="navDark"

>

<Container>

<NavbarBrand href="#">BCI Browser</NavbarBrand>

<NavbarToggler onClick={this.toggle} />

<Collapse isOpen={this.state.isOpen} navbar>

<Nav className="ml-auto" navbar>

<NavItem>

<NavLink href="#about">About</NavLink>

</NavItem>

<NavItem>

<NavLink href="#packageBody">P300 Speller</NavLink>

</NavItem>

<NavItem>

<NavLink href="#servicesBody">Services</NavLink>

</NavItem>

<NavItem>

<NavLink href="#contactBody">Contact</NavLink>

</NavItem>

<Button color="success">Login</Button>{" "}

</Nav>

</Collapse>

</Container>

</Navbar>

</div>

);

}

}

export default NavbarMain;

//**REACTJS code for Footer**

import React, { Component } from "react";

import { Container, Row, Col, Button } from "reactstrap";

import "../App.css";

import { ReactComponent as Facebook } from "../assets/svg/facebook.svg";

import { ReactComponent as Twitter } from "../assets/svg/twitter.svg";

import { ReactComponent as Linkedin } from "../assets/svg/linkedin.svg";

import { ReactComponent as Instagram } from "../assets/svg/instagram.svg";

class Footer extends Component {

render() {

return (

<div className="subComponent-lg" id="footerBody">

<Container>

<header className="headerTitle text-center">

<p>

<b>CAN'T WAIT TO GET INVOLVED</b>

</p>

<p>

<b>CONNECT WITH US NOW</b>

</p>

</header>

<footer className="svg-group text-center">

<Row>

<Col md="3" xs="6">

<div className="svg-card-3">

<a href="#">

<Facebook width="50" height="55" strokeWidth="1" />

</a>

</div>

</Col>

<Col md="3" xs="6">

<div className="svg-card-3">

<a href="#">

<Twitter width="55" height="55" strokeWidth="1" />

</a>

</div>

</Col>

<Col md="3" xs="6">

<div className="svg-card-3">

<a href="#">

<Linkedin width="55" height="55" strokeWidth="1" />

</a>

</div>

</Col>

<Col md="3" xs="6">

<div className="svg-card-3">

<a href="#">

<Instagram width="55" height="55" strokeWidth="1" />

</a>

</div>

</Col>

</Row>

<hr />

<br />

<p>copyright 2021 | BRAIN-COMPUTER INTERFACE BROWSER</p>

</footer>

</Container>

</div>

);

}

}

export default Footer;

**Knowledge Repository using POSTGRESQL**

Bulk loading is the quickest way to import large amounts of data into a PostgreSQL database. There are various ways to facilitate large-scale imports, and many different ways to scale are also available. This post will show you how to use some of these tricks, and explain how fast importing works. You can use this knowledge to optimize data warehousing or any other data-intensive workload.

There are several things to take into consideration in order to speed up bulk loading of massive amounts of data using PostgreSQL:

* INSERT vs. COPY
* Optimizing checkpoints
* Logged vs. unlogged tables
* Recreating indexes
* Enabled and disabled triggers
* Improving column order and space consumption

### **INSERT vs. COPY**

The first thing to consider is that COPY is usually a LOT better than plain inserts. The reason is that INSERT has a lot of overhead. People often ask: What kind of overhead is there? What makes COPY so much faster than INSERT? There are a variety of reasons: In the case of INSERT, every statement has to check for locks, check for the existence of the table and the columns in the table, check for permissions, look up data types and so on. In the case of COPY, this is only done once, which is a lot faster. Whenever you want to write large amounts of data, data COPY is usually the way to go.

### **Adjusting checkpoints for faster bulk loading**

The PostgreSQL configuration does have an impact on bulk loading performance. There are many configuration parameters which are vital to database performance, and loading in particular. However, I explicitly want to focus your attention on checkpoint and I/O performance. If you want to load billions of rows, I/O is king. There are various angles to approach the topic:

* Reduce the amount of data written
  + Make tables smaller if possible (column order)
  + Reduce the amount of WAL written
* Write data more efficiently
  + Longer checkpoint distances
  + Better I/O scheduling

The following settings are important:

* max\_wal\_size: Maximum amount of WAL to be created (soft limit)
* checkpoint\_completion\_target: Control checkpoint behavior

In general it is a REALLY good idea to stretch checkpoints by a significant amount. Setting this value to 100 or 200 GB in case of bulk-load intense workloads is definitely not out of scope.

Keep in mind that increased checkpoint distances DO NOT put your server at risk. It merely affects the way PostgreSQL writes data. Also keep in mind that more disk space will be consumed and recovery might take longer, in case of a crash.

If you want to learn more about checkpointing, check out [this article about reducing the amount of WAL written.](https://www.cybertec-postgresql.com/en/checkpoint-distance-and-amount-of-wal/)

### **Tooling for bulk loading**

If what you have seen so far is still not enough, we can recommend some tools to improve bulk loading even more. The following tools can be recommended:

* [PGLoader](https://github.com/dimitri/pgloader)
* [pg\_bulkload](https://github.com/ossc-db/pg_bulkload)

Both tools are very well known and widely used. You can use them safely.

**3-- Visualization of Data:**

The magnitude of brain wave signal is determined with an EEG(electroencephalogram) measuring and processing unit, produced by NeuroSky. The read, the conversion and the various visualization of the measured and processed signals are implemented in the developed software.

A Windows Forms Application has been developed to evaluate and visualize the brain wave data of the MindFlex EEG headset.

The following requirements have been considered regarding the **visualization** and evaluation software:

• it should be able to receive data sent through serial

communication by the EEG headset;

• it can check if any data transfer errors have

emerged during transmission;

• it stores data in adequate format and structure for

further procession;

• it illustrates processed data in a column chart for

evaluation;

• it can execute data evaluation faster than the data

sending speed of the headset;

• in case of the PC software, the investigation of sig-

nals and display of the measured and processed da-

ta alternation has also been achieved to be able to

observe changes in the brain wave signals.

For the software development Microsoft Visual Studio was applied. This development environment supports modern object-oriented programming on Windows operating system.

The developed program can be divided into four main functions:

the EEG device data communication,

the data processing and analysis,

the visualization;

furthermore, the control communication with nodes.

The EEG device data communication function reads, converts the data received

from the headset through serial connection. The data processing and analysis function realize the FFT spectrum analysis, the visualization part displays the spectrum by a column chart and the time function of the brain signal by a

line chart.

The control communication part realizes the control interactions with other nodes using VSCP protocol.

The user interface can be seen in Figure 4. The source code contains three classes: one of them is

the so-called BrainWaveReaderForm, which deals with

events and derives from the Form class which derives

from the source of all classes named Object.

BrainWaveReaderForm is responsible for data processing and control communication between other nodes. The second class is called the FormGraphics class and is responsible for data visualization and is in connection with the BrainWaveReaderForm class. The third class is the SerialPortManag-

er, which manages the EEG serial port communication.

In Figure 5, you can see the flowchart illustrating the main

functions of the program

