JavaScript-Code

Seawater-Freshwater Interface JavaScript Implementation

Code Review

```
// Ghyben-Herzberg formula function
function ghybenHerzberg(x, q, pf, ps, K) {
  Return Math.sqrt((2 * pf * q * x) / ((ps - pf) * K));
}

// Glover formula function
function glover(x, q, pf, ps, K) {
  Return Math.sqrt((2 * pf * q * x) / ((ps - pf) * K) + Math.pow((pf * q /
(ps - pf) / K), 2));
}

// Rumer Jr & Harleman formula function
function rumerHarleman(x, q, pf, ps, K) {
  Return Math.sqrt((2 * pf * q * x) / ((ps - pf) * K) + 0.55 * Math.pow((pf
* q / (ps - pf) / K), 2));
}

// Verruijt formula function
function verruijt(x, q, K, B) {
  Return Math.sqrt(Math.pow((q / B / K), 2) * (1 - B) / (1 + B) + 2 * q * x
/ (B * K * (1 + B)));
}

// Function to get user data from the HTML elements
function GetUserDate() {
  // Fetch the values from the HTML elements and convert them to the
appropriate units
  Let Q = +document.getElementById('Q').value * 1000000;  // Flow rate
  Let K = +document.getElementById('K').value * 100;  // Hydraulic
conductivity
  Let pf = +document.getElementById('pf').value;  // Fluid density
  Let ps = +document.getElementById('ps').value;  // Solid density
  Let hs = +document.getElementById('hs').value * 100;  // Solid height
  Let hf = 0;  // Fluid height
  Let L = +document.getElementById('L').value * 100;  // Length
  Let step = +document.getElementById('step').value;  // Step size
  Let formula = +document.getElementById('formula').value;  // Selected
formula
```

```
  Return { Q, K, pf, ps, hs, hf, L, step, formula };
}

// Function to get the z value based on the selected formula
function GetZValue(Data, x) {
  Let q = Data.Q / 8;
  Let B = (Data.ps - Data.pf) / Data.pf;
  Let kstar = Data.K * B;

  // Switch case to select the formula based on the user's choice
  Switch (Data.formula) {
    Case Enumerator.GhybenHerzberg:
      Return ghybenHerzberg(x, q, Data.pf, Data.ps, Data.K) / 100;
    Case Enumerator.Glover:
      Return glover(x, q, Data.pf, Data.ps, Data.K) / 100;
    Case Enumerator.RumerHarleman:
      Return rumerHarleman(x, q, Data.pf, Data.ps, Data.K) / 100;
    Case Enumerator.Verruijt:
      Return verruijt(x, q, Data.K, B) / 100;
    Default:
      Return 0;
  }
}

// Function to update the table on the UI with the calculated values
function UpdateTable(table, xtoe, Data) {
  Let rowsCount = Math.ceil(xtoe / Data.step);
  Let data = [];
  For (let I = 0; I < rowsCount; i++) {
    Let x = I * Data.step;
    Let z;
    If (x < xtoe) {
      Z = GetZValue(Data, x * 100);
    } else {
      Break;
    }
    Data.push([x.toFixed(3), z]);
  }
  Data.push([xtoe.toFixed(4), GetZValue(Data, xtoe * 100)]);
  Table.rows().remove().draw();
  Table.rows.add(data).draw(false);
}

// Function to get the x value at the toe
function GetXToeValues(Data) {
  Let step = 0.0001;
  Let z = 0;
  Let x = 0;
  While (z < Data.hs / 100) {
```

```
    Z = GetZValue(Data, (x * 100));
    X += step;
  }
  Return x - 2 * step;
}

Let myChartInstance;

// Function to load the chart with the calculated values
function LoadChart(Data) {
  Const ctx = document.getElementById('myChart');

  If (myChartInstance) {
    myChartInstance.destroy();
  }

  Ctx.classList.remove('d-none');

  // Extract data from the table
  Let tableData = $('#myTable').DataTable().rows().data().toArray();
  Let labels = tableData.map(row => row[0]);  // x values
  Let data = tableData.map(row => row[1]);  // z values

  // Define formula names
  Const formulaNames = [
    "Ghyben-Herzberg",
    "Glover",
    "Rumer-Harleman",
    "Verruijt"
  ];

  // Create a new chart instance and assign it to the global variable
  myChartInstance = new Chart(ctx, {
    type: 'line',
    data: {
      labels: labels,
      datasets: [{
        label: formulaNames[Data.formula],
        data: data,
        borderWidth: 1
      }]
    },
    Options: {
      Scales: {
        X: {
          Reverse: true,  // Reverse the x-axis
          Position: 'top'  // Position the x-axis at the top
        },
        Y: {
```

```
          beginAtZero: true,
          reverse: true,  // Reverse the x-axis
          position: 'right'  // Position the x-axis at the top
        }
      }
    }
  });
}

// Enumerator for the formulas
Enumerator = {
  GhybenHerzberg: 0,
  Glover: 1,
  RumerHarleman: 2,
  Verruijt: 3
}

// Function to calculate the values, update the table and load the chart
Async function Calculate(table) {
  Let Data = GetUserDate();
  Let xtoe = await GetXToeValues(Data);
  UpdateTable(table, xtoe, Data);
  LoadChart(Data);
}

// Event listener for the 'calculate' button
Document.addEventListener('DOMContentLoaded', function () {
  Document.getElementById('calculate').addEventListener('click', function ()
{
    Calculate(table);
  });

  // Initialize the table
  Let table = new DataTable('#myTable', {
    Columns: [
      { label: 'x', name: 'x', title: 'x (m)' },
      { label: 'z', name: 'z', title: 'z (m)' },
    ]
  });
});
```

Code Analysis and Explanation

Step-by-Step Explanation

Step 1: Adding Event Listeners and Initializing the Table

The code begins by setting up event listeners and initializing the HTML table using the DataTable library. This occurs when the DOM content is fully loaded.

1. **Event Listener for 'calculate' Button**:
   - The script attaches an event listener to the 'calculate' button. This triggers the `Calculate` function upon a button click.
   - The `Calculate` function retrieves user input values, processes the calculations, and updates the user interface accordingly.
2. **Initializing the DataTable**:
   - The table is initialized using the DataTable library with columns labeled 'x (m)' and 'z (m)'. These columns will display the x and z values generated by the calculations.

```
document.addEventListener('DOMContentLoaded', function () {
  document.getElementById('calculate').addEventListener('click', function ()
{
    Calculate(table);
  });

  // Initialize the table
  let table = new DataTable('#myTable', {
    columns: [
      { label: 'x', name: 'x', title: 'x (m)' },
      { label: 'z', name: 'z', title: 'z (m)' },
    ]
  });
});
```

Step 2: Retrieving User Input Data

Upon clicking the 'calculate' button, the `Calculate` function is invoked. This function calls `GetUserData` to collect and convert user inputs from the HTML form elements.

```
function GetUserData() {
  let Q = +document.getElementById('Q').value * 1000000;
  let K = +document.getElementById('K').value * 100;
  let pf = +document.getElementById('pf').value;
  let ps = +document.getElementById('ps').value;
  let hs = +document.getElementById('hs').value * 100;
  let hf = 0;
  let L = +document.getElementById('L').value * 100;
  let step = +document.getElementById('step').value;
  let formula = +document.getElementById('formula').value;
  return { Q, K, pf, ps, hs, hf, L, step, formula };
}
```

Step 3: Calculating x at the Interface Toe

The `GetXToeValues` function calculates the x-coordinate at the toe of the seawater-freshwater interface. This function iteratively calculates z-values for increasing x-values until z is approximately equal to the solid height (`hs`) divided by 100 (converted to meters).

```
function GetXToeValues(Data) {
  let step = 0.0001;
  let z = 0;
  let x = 0;
  while (z < Data.hs / 100) {
    z = GetZValue(Data, x * 100);
    x += step;
  }
  return x - 2 * step;
}
```

Step 4: Updating the Table with Calculated Values

The `UpdateTable` function populates the HTML table with the calculated x and z values. This function iterates through the x-values up to the toe (`xtoe`), calculates the corresponding z-values using the selected formula, and adds them to the table.

```
function UpdateTable(table, xtoe, Data) {
  let rowsCount = Math.ceil(xtoe / Data.step);
  let data = [];
  for (let i = 0; i < rowsCount; i++) {
    let x = i * Data.step;
    let z;
    if (x < xtoe) {
      z = GetZValue(Data, x * 100);
    } else {
      break;
    }
    data.push([x.toFixed(3), z]);
  }
  data.push([xtoe.toFixed(4), GetZValue(Data, xtoe * 100)]);
  table.rows().remove().draw();
  table.rows.add(data).draw(false);
}
```

Step 5: Loading the Chart with Calculated Values

Finally, the `LoadChart` function generates a visual representation of the calculated values using Chart.js. It extracts x and z values from the table and plots them on a line chart. The chart configuration includes labels and a dataset representing the calculated values.

```
function LoadChart(Data) {
  const ctx = document.getElementById('myChart');

  if (myChartInstance) {
    myChartInstance.destroy();
  }

  ctx.classList.remove('d-none');

  // Extract data from the table
  let tableData = $('#myTable').DataTable().rows().data().toArray();
  let labels = tableData.map(row => row[0]);  // x values
  let data = tableData.map(row => row[1]);  // z values

  // Define formula names
  const formulaNames = [
    "Ghyben-Herzberg",
    "Glover",
    "Rumer-Harleman",
    "Verruijt"
  ];

  // Create a new chart instance and assign it to the global variable
  myChartInstance = new Chart(ctx, {
    type: 'line',
    data: {
      labels: labels,
      datasets: [{
        label: formulaNames[Data.formula],
        data: data,
        borderWidth: 1
      }]
    },
    options: {
      scales: {
        x: {
          reverse: true,  // Reverse the x-axis
          position: 'top'  // Position the x-axis at the top
        },
        y: {
          beginAtZero: true,
          reverse: true,  // Reverse the y-axis
          position: 'right'  // Position the y-axis at the right
        }
      }
    }
  });
}
```

Impact of Sea Level Rise (SLR) on Toe of SW-FW Interface JavaScript Implementation

Code Review

```
function GetUserInput()
{
    let q = +document.getElementById('q-2').value;
    let W = +document.getElementById('W-2').value/365000;
    let pf = +document.getElementById('pf-2').value
    let ps = +document.getElementById('ps-2').value;
    let K = +document.getElementById('K-2').value;
    let L = +document.getElementById('L-2').value;
    let Zo = +document.getElementById('Zo-2').value
    let Δz = +document.getElementById('Δz-2').value
    let S = +document.getElementById('S-2').value
    let hb = +document.getElementById('hb-2').value;
    let δ = (ps - pf) / pf;
    let formula = document.getElementById('formula-2').value;
    return { q, W, δ, K, L, Zo, Δz, S, hb, formula };
}

Enumerator = {
    CFB: 0,
    CHB: 1
}

function CFBXTCalculte(Data)
{
    let part1 = (Data.q / Data.W) + Data.L;
    let part2 = ((Data.K * Data.δ * (1 + Data.δ) * Math.pow(Data.Zo, 2)) /
(Data.W));
    return part1 - Math.sqrt(Math.pow(part1, 2) - part2);
}

function CFBXDashTCalculte(Data) {
    let part1 = (Data.q / Data.W) + Data.L - (Data.Δz / Data.S);
    let part2 = ((Data.K * Data.δ * (1 + Data.δ) * Math.pow(Data.Zo +
Data.Δz, 2)) / (Data.W));
    return part1 - Math.sqrt(Math.pow(part1, 2) - part2) + (Data.Δz /
Data.S);
}

function CHBXTCalculte(Data) {
    let qPart1 = (Data.K * (Math.pow(Data.hb + Data.Zo, 2) - (1 + Data.δ) *
Math.pow(Data.Zo, 2))) / (2 * Data.L);
    let qPart2 = Data.W * Data.L / 2;
    let q = qPart1 - qPart2;
```

```
    Data.q = q;
    return CFBXTCalculte(Data);
}

function CHBXDashTCalculte(Data) {
    let qPart1 = (Data.K * (Math.pow(Data.hb + Data.Zo, 2) - (1 + Data.δ) *
Math.pow(Data.Zo + Data.Δz, 2))) / (2 * (Data.L - Data.Δz / Data.S));
    let qPart2 = Data.W * (Data.L - Data.Δz / Data.S) / 2;
    let q = qPart1 - qPart2;
    Data.q = q;
    return CFBXDashTCalculte(Data);
}

function calculateXT(Data){
    if(Data.formula == Enumerator.CFB)
    {
        let X = CFBXTCalculte(Data);
        document.getElementById('XTR-2').value = X;
    }
    else if(Data.formula == Enumerator.CHB)
    {
        let X = CHBXTCalculte(Data);
        document.getElementById('XTR-2').value = X;
    }
}

function calculateXTDash(Data){
    if(Data.formula == Enumerator.CFB)
    {
        let XDash = CFBXDashTCalculte(Data);
        document.getElementById("XT'R-2").value = XDash;
    }
    else if(Data.formula == Enumerator.CHB)
    {
        let XDash = CHBXDashTCalculte(Data);
        document.getElementById("XT'R-2").value = XDash;
    }
}

document.addEventListener('DOMContentLoaded', function () {
    document.getElementById('formula-2').addEventListener('change', function
() {
        let formula = document.getElementById('formula-2').value;
        if (formula == Enumerator.CFB) {
            document.getElementById('qcontainer').classList.remove('d-
none');
            document.getElementById('hbcontainer').classList.add('d-none');
        }
        else if (formula == Enumerator.CHB) {
```

```
            document.getElementById('qcontainer').classList.add('d-none');
            document.getElementById('hbcontainer').classList.remove('d-
none');
        }
    });

    document.getElementById('calculate-XT-2').addEventListener('click',
function () {
        let Data = GetUserInput();
        calculateXT(Data);
    });

    document.getElementById("calculate-X'T-2").addEventListener('click',
function () {
        let Data = GetUserInput();
        calculateXTDash(Data);
    });
});
```

Code Analysis and Explanation

Step-by-Step Explanation

Step 1: Adding Event Listeners and Managing UI Interactions

The JavaScript code initializes by setting up event listeners to handle user interactions. These listeners respond to user actions like selecting formulas and clicking calculation buttons. This setup occurs when the DOM content is fully loaded.

1. **Event Listener for Formula Selection**:
    - An event listener is attached to the formula dropdown menu. Depending on the selected formula, the relevant input fields are shown or hidden. This ensures users input only the necessary values for their chosen formula.
2. **Event Listeners for 'Calculate XT' and 'Calculate XT'' Buttons**:
    - These buttons trigger the computation of the interface's x-coordinate (`XT`) and its adjusted value (`XT'`) considering sea level rise. The respective functions `calculateXT` and `calculateXTDash` are invoked upon clicking.

```
document.addEventListener('DOMContentLoaded', function () {
    document.getElementById('formula-2').addEventListener('change', function
() {
        let formula = document.getElementById('formula-2').value;
        if (formula == Enumerator.CFB) {
            document.getElementById('qcontainer').classList.remove('d-
none');
            document.getElementById('hbcontainer').classList.add('d-none');
```

```
        } else if (formula == Enumerator.CHB) {
            document.getElementById('qcontainer').classList.add('d-none');
            document.getElementById('hbcontainer').classList.remove('d-
none');
        }
    });

    document.getElementById('calculate-XT-2').addEventListener('click',
function () {
        let Data = GetUserInput();
        calculateXT(Data);
    });

    document.getElementById("calculate-X'T-2").addEventListener('click',
function () {
        let Data = GetUserInput();
        calculateXTDash(Data);
    });
});
```

Step 2: Retrieving User Input

The GetUserInput function collects and processes user inputs from the HTML elements. It converts the values into appropriate units and formats them into an object. This object includes all necessary parameters for the calculations, such as flow rate, density, and hydraulic conductivity.

```
function GetUserInput() {
    let q = +document.getElementById('q-2').value;
    let W = +document.getElementById('W-2').value / 365000;
    let pf = +document.getElementById('pf-2').value;
    let ps = +document.getElementById('ps-2').value;
    let K = +document.getElementById('K-2').value;
    let L = +document.getElementById('L-2').value;
    let Zo = +document.getElementById('Zo-2').value;
    let Δz = +document.getElementById('Δz-2').value;
    let S = +document.getElementById('S-2').value;
    let hb = +document.getElementById('hb-2').value;
    let δ = (ps - pf) / pf;
    let formula = document.getElementById('formula-2').value;
    return { q, W, δ, K, L, Zo, Δz, S, hb, formula };
}
```

Step 3: Calculating XT and XT' Values

The code includes functions to calculate the x-coordinate at the interface toe (XT) and its modified value (XT') considering sea level rise. These functions apply specific hydrodynamic

formulas based on the user's selected formula.

- CFB (Constant Flux Boundary) Calculation:
  - The CFBXTCalculte function computes the XT value using the Constant Flux Boundary formula.
  - The CFBXDashTCalculte function calculates the XT' value under sea level rise conditions.

```
function CFBXTCalculte(Data) {
    let part1 = (Data.q / Data.W) + Data.L;
    let part2 = ((Data.K * Data.δ * (1 + Data.δ) * Math.pow(Data.Zo, 2)) /
(Data.W));
    return part1 - Math.sqrt(Math.pow(part1, 2) - part2);
}

function CFBXDashTCalculte(Data) {
    let part1 = (Data.q / Data.W) + Data.L - (Data.Δz / Data.S);
    let part2 = ((Data.K * Data.δ * (1 + Data.δ) * Math.pow(Data.Zo +
Data.Δz, 2)) / (Data.W));
    return part1 - Math.sqrt(Math.pow(part1, 2) - part2) + (Data.Δz /
Data.S);
}
```

- CHB (Constant Head Boundary) Calculation:
  - The CHBXTCalculte function computes the XT value using the Constant Head Boundary formula.
  - The CHBXDashTCalculte function calculates the XT' value considering the impact of sea level rise.

```
function CHBXTCalculte(Data) {
    let qPart1 = (Data.K * (Math.pow(Data.hb + Data.Zo, 2) - (1 + Data.δ) *
Math.pow(Data.Zo, 2))) / (2 * Data.L);
    let qPart2 = Data.W * Data.L / 2;
    let q = qPart1 - qPart2;
    Data.q = q;
    return CFBXTCalculte(Data);
}

function CHBXDashTCalculte(Data) {
    let qPart1 = (Data.K * (Math.pow(Data.hb + Data.Zo, 2) - (1 + Data.δ) *
Math.pow(Data.Zo + Data.Δz, 2))) / (2 * (Data.L - Data.Δz / Data.S));
    let qPart2 = Data.W * (Data.L - Data.Δz / Data.S) / 2;
    let q = qPart1 - qPart2;
    Data.q = q;
    return CFBXDashTCalculte(Data);
}
```

Step 4: Updating Interface Values Based on Selected Formula

The functions calculateXT and calculateXTDash determine the x-coordinates (XT and XT') based on the selected formula. They update the corresponding HTML elements with the computed values.

```
function calculateXT(Data) {
    if (Data.formula == Enumerator.CFB) {
        let X = CFBXTCalculte(Data);
        document.getElementById('XTR-2').value = X;
    } else if (Data.formula == Enumerator.CHB) {
        let X = CHBXTCalculte(Data);
        document.getElementById('XTR-2').value = X;
    }
}

function calculateXTDash(Data) {
    if (Data.formula == Enumerator.CFB) {
        let XDash = CFBXDashTCalculte(Data);
        document.getElementById("XT'R-2").value = XDash;
    } else if (Data.formula == Enumerator.CHB) {
        let XDash = CHBXDashTCalculte(Data);
        document.getElementById("XT'R-2").value = XDash;
    }
}
```