**Web RPD Documentation**

**Overview**

The Web RPD (Removable Partial Denture) application is a comprehensive web-based tool designed to visualize and manage dental models, specifically focusing on the creation and modification of removable partial dentures. The application leverages various libraries and APIs to provide a rich 3D visualization experience, allowing users to interact with dental models, view different data layers, and control various visualization parameters.

**Libraries and Dependencies**

The application primarily uses the following libraries:

* **THREE.js**: A JavaScript library used to create and display animated 3D graphics in the browser.
* **OrbitControls** and **TrackballControls**: Extensions of THREE.js to allow for interactive control of the camera.
* **STLMeshLoader**: A custom loader for processing STL files.
* **OFFLoader**: A custom loader for processing OFF files.
* **ApiClient**: A custom class to handle API requests.

**Index.js**

**Initialization**

The script begins by importing necessary libraries and initializing essential variables and objects, such as the scene, camera, and renderer. It sets up the environment for rendering 3D models and handling user interactions.

**URL Parameters**

* **id**: The case ID for the dental model to be loaded.
* **close**: A flag indicating whether to load a closed mesh representation.

The paramValue is decrypted if necessary for security purposes.

**Scene and Camera Setup**

* **Scene**: A new THREE.js scene is created.
* **Camera**: An orthographic camera is initialized with specific bounds and added to the scene.

**Material Setup**

Different materials are defined for the meshes to handle various rendering properties:

* **material**: Used for general meshes.
* **materialsurface** and **materialsurface\_non\_metal**: Used for surface meshes with different properties (metallic and non-metallic).

**API Client Initialization**

An instance of the ApiClient is created to handle communication with the backend API.

**Data Fetching**

Data is fetched asynchronously using the ApiClient. This includes:

* Case information (creation date, case ID, last updated).
* Thumbnail images for different slots.
* Undercut heatmap values.
* Mesh data (both parameterized and surface meshes).

**Mesh Processing**

The application supports two main types of meshes:

* **OFF files**: Loaded using the OFFLoader class.
* **STL files**: Loaded using the STLMeshLoader class if the OFF files cannot be processed.

The meshes are parsed, transformed, and added to the scene. The application also checks for closed meshes and provides a button to switch between original and closed mesh views.

**UI Elements**

* **Textboxes**: Display creation date, case ID, and last updated information.
* **Thumbnail Buttons**: Display thumbnail images and allow for fullscreen view on click.
* **Visibility and Transparency Controls**: Added to manage mesh visibility and transparency dynamically.
* **Reset Button**: Resets the camera view to the original state.

**Event Listeners**

* **Window Resize**: Adjusts the camera and renderer size when the window is resized.
* **Mouse Move**: Tracks mouse position for interactive elements.
* **Button Clicks**: Handles various button clicks for changing mesh views and displaying images.

**Rendering**

The application uses an animation loop to render the scene continuously. The camera and controls are updated within this loop to ensure smooth interaction.

**Utility Functions**

* **createTextbox**: Creates and styles textboxes for displaying information.
* **unixToHumanReadable**: Converts Unix timestamps to human-readable dates.
* **displayFullScreenImage**: Displays images in fullscreen mode with proper aspect ratio handling.

**Logout**

The application logs out the user after loading the necessary data by calling the /user/logout endpoint.

**ApiClient Class Documentation**

**Overview**

The ApiClient class is a core component of the WebRPD system, responsible for managing HTTP requests and handling responses. It includes functionality for posting data to endpoints, managing progress updates, and processing login operations. The login function is to be called before every fetch request to ensure that external user’s logout would not affect the operations.

**ApiClient Class**

**Constructor**

constructor(baseUrl)

* **Parameters:**
  + baseUrl: The base URL for API requests.
* **Description:** Initializes the ApiClient instance with the given baseUrl.

**post**

async post(endpoint, data, test, what)

* **Parameters:**
  + endpoint: The API endpoint to which the request is sent.
  + data: The data to be sent in the body of the request.
  + test: A boolean flag to determine the return value on success.
  + what: Optional string to describe the download process.
* **Returns:**
  + On success, returns the parsed JSON response if test is false or true if test is true.
  + On failure, returns 'stl' for specific HTTP statuses or throws an error for other statuses.
* **Description:**
  + Sends a POST request to the specified endpoint.
  + Handles and displays download progress via a progress bar and percentage indicator.
  + Calculates and displays download speed in megabytes per second (MB/s).

**Error Handling**

* If the response status is 500 or 404, the method returns 'stl'.
* For other errors, an Error is thrown with the status of the response.

**Progress Display**

* **Progress Bar:** A visual progress bar is created to show the download progress.
* **Percentage:** A span element displays the current download percentage.
* **Download Speed:** Displays the download speed in MB/s, updated in real-time.

**login Function**

async function login()

* **Returns:** The JSON response from the login API.
* **Description:**
  + Sends a POST request to the login endpoint with predefined credentials and machine information.
  + Handles login authentication and retrieves session data.

**Error Handling**

* If the login request fails, an error is logged and rethrown for further handling.

**HTML Elements Created**

During the file download process, the following HTML elements are dynamically created and styled:

* **Container:** A div element that acts as a wrapper for the progress display elements.
* **Progress Bar:** A progress element indicating the download progress.
* **Percentage:** A span element showing the download percentage.
* **Display Box:** A div element used to display download speed and other status messages.

**Styling Details**

* **Container:** Centers all elements vertically and horizontally in the viewport.
* **Progress Bar:** Styled to take up 80% of the container's width.
* **Percentage:** Positioned below the progress bar with a margin for separation.
* **Display Box:** Contains status messages and is styled with padding, border radius, and a transparent background.

**OFFLoader Class Documentation**

**Overview**

The OFFLoader class is a part of the Web RPD and is used to load and process 3D models from OFF (Object File Format) files. This class leverages the three.js library to handle 3D geometry and materials, providing a means to visualize and interact with 3D models within a web-based environment.

**Class Definition**

**Import Statement:**

import \* as THREE from "../node\_modules/three/build/three.module.js";

**Class Definition:**

export class OFFLoader {

constructor(material, submaterial) { this.material = material; if (typeof submaterial != 'undefined') { this.submaterial = submaterial; } }

**Constructor**

* **Parameters:**
  + material (THREE.Material): The default material to be used for rendering the geometry.
  + submaterial (THREE.Material, optional): An additional material for specialized rendering needs.
* **Description:** Initializes the OFFLoader instance with the specified materials. If a submaterial is provided, it will be stored for later use.

**Methods**

**parse(data, surface, check)**

* **Parameters:**
  + data (string): The OFF file data as a string.
  + surface (object): An object containing additional properties like occlusion\_values and surveying\_values for advanced visualization.
  + check (boolean): A flag indicating whether to apply default color settings.
* **Returns:**
  + An array where:
    - The first element is a THREE.Mesh instance representing the loaded 3D model.
    - The second element is an array of geometries and materials used.
* **Description:** Parses the OFF file data and processes it into a THREE.Mesh object with associated materials. The method handles various types of OFF file data and additional surface properties to enhance the 3D model visualization.
* **Key Features:**
  + **File Validation:**
    - Checks if the file starts with "OFF" to ensure it is a valid OFF file.
  + **Geometry Creation:**
    - Extracts vertex and face data to create a THREE.BufferGeometry object.
    - Supports only triangular faces. Non-triangular faces are warned but ignored.
  + **Default Color Setting:**
    - If check is true, assigns a default color (light brown) to the geometry.
  + **Occlusion Color Handling:**
    - If surface contains occlusion\_values, it processes these values to adjust the vertex colors of the geometry.
  + **Undercut Value Handling:**
    - If surface contains surveying\_values, it processes these values to further adjust the vertex colors of the geometry.
  + **Mesh Creation:**
    - Creates a THREE.Mesh using either the default or provided submaterial.
    - Includes the geometry and material(s) in the returned array.

**Example Usage**

const loader = new OFFLoader(defaultMaterial, subMaterial);

fetch('path/to/model.off') .then(response => response.text()) .then(data => { const [mesh, materials] = loader.parse(data, surfaceData, true); scene.add(mesh); // Add the mesh to the scene });

**Notes**

* Ensure that the OFF file data is correctly formatted and contains valid face and vertex data.
* The parse method is designed to handle additional data for advanced visualization. Ensure that the surface object is correctly populated with occlusion\_values and/or surveying\_values if needed.

**STLMeshLoader Class Documentation**

**Overview**

The STLMeshLoader class is designed to facilitate the loading and processing of STL (stereolithography) files for 3D rendering. It uses the Three.js library for handling 3D geometries and materials. The class includes functionality for processing STL files, merging vertices to optimize geometry, and applying different color attributes based on input data.

**Dependencies**

* **Three.js**: A JavaScript library for 3D graphics.
* **STLLoader**: A Three.js loader specifically for STL files.

**Class Definition**

**Import Statement:**

import { STLLoader } from '../node\_modules/three/examples/jsm/loaders/STLLoader.js'; import \* as THREE from 'three';

**Class Definition:**

class STLMeshLoader { constructor(material) { this.material = material; }

load(data, surface) { // Method Implementation } }

export { STLMeshLoader };

**Constructor**

**STLMeshLoader(material)**

* **Parameters:**
  + material: A Three.js material object that will be used to render the mesh.
* **Description:** Initializes the STLMeshLoader instance with a material for rendering.

**Methods**

**load(data, surface)**

* **Parameters:**
  + data: The STL file data in a format that the STLLoader can parse.
  + surface: An object containing additional surface data which may include occlusion\_values and/or surveying\_values.
* **Returns:**
  + An array containing:
    - A THREE.Mesh object with the processed geometry and the provided material.
    - An array of THREE.BufferGeometry objects with different color attributes.
* **Description:** Loads and processes the STL file data, applies color attributes based on the provided surface data, and creates a THREE.Mesh object. The method performs the following steps:
  + **Loading Geometry:**
    - Utilizes STLLoader to parse the STL data and create an initial geometry.
  + **Merging Vertices:**
    - Calls the mergeVertices function to remove duplicate vertices and optimize the geometry.
  + **Color Application:**
    - Sets default colors (light brown) for the geometry.
    - If occlusion\_values are provided in the surface, processes these values to update vertex colors.
    - If surveying\_values are provided, processes these as well to further update vertex colors.
  + **Mesh Creation:**
    - Creates a THREE.Mesh using the processed geometry and the provided material.

**Utility Functions**

**mergeVertices(geometry)**

* **Parameters:**
  + geometry: The THREE.BufferGeometry object to optimize.
* **Returns:**
  + A new THREE.BufferGeometry object with merged vertices.
* **Description:** Optimizes the geometry by merging vertices that are close to each other. This function helps to reduce the number of vertices and indices in the geometry, improving performance and rendering efficiency. The function does the following:
  + **Vertex Mapping:**
    - Maps vertices using a threshold to determine when vertices should be considered the same.
  + **Geometry Creation:**
    - Constructs a new THREE.BufferGeometry object with the merged vertices and indices.

**Notes**

* The color values are set to a default light brown unless overridden by the occlusion\_values or surveying\_valuesfrom the surface object.
* The mergeVertices function uses a simple rounding method to determine vertex uniqueness, which may need adjustment based on the specific requirements of the application.

**WebRPD Crypt.js Functions and Features**

**Overview**

The crypt.js module provides functionality for encoding and decoding data using Base64 encoding and XOR encryption/decryption techniques. The primary function, lol, demonstrates how these methods can be used in practice.

**Functions and Features**

**For reference**

**Current key =** PgrJrkwpeG9pd

Decrypted id = 797

Encrypted id = OAJBLhkNOQoOdFADDmMBGQ

**1. decodeStringToNumber(encodedString)**

* **Purpose:** Converts a Base64 encoded string into a number.
* **Parameters:**
  + encodedString (String): The Base64 encoded string to be decoded.
* **Returns:**
  + (Number): The decoded number.
* **Implementation:**
  + Uses atob() to decode the Base64 string to its original format and parseInt() to convert the decoded string into a number.

**2. xorDecrypt(encryptedBase64, key)**

* **Purpose:** Decrypts a Base64 encoded, XOR encrypted string using a specified key.
* **Parameters:**
  + encryptedBase64 (String): The Base64 encoded string that has been XOR encrypted.
  + key (String): The key used for XOR decryption.
* **Returns:**
  + (String): The decrypted text.
* **Implementation:**
  + Replaces characters in the Base64 string to reverse URL-safe encoding.
  + Uses atob() to decode the Base64 string to its original encrypted text.
  + Applies XOR decryption on each character of the encrypted text using the provided key.

**3. lol(things)**

* **Purpose:** Demonstrates how to decode a number using XOR decryption and Base64 encoding.
* **Parameters:**
  + things (String): The input string to be decoded.
* **Returns:**
  + (Number): The decoded number obtained from the XOR decryption and Base64 decoding process.
* **Implementation:**
  + Calls xorDecrypt() with the input string and a hardcoded key.
  + Uses decodeStringToNumber() to convert the decrypted string into a number.
  + Returns the decoded number.

**Additional Functions (Commented Out)**

* **encodeNumberToString(number)** (Commented Out)
  + **Purpose:** Encodes a number into a Base64 string.
  + **Parameters:**
    - number (Number): The number to be encoded.
  + **Returns:**
    - (String): The Base64 encoded string representing the number.
  + **Implementation:**
    - Converts the number to a string and then encodes it using btoa().
* **xorEncrypt(text, key)** (Commented Out)
  + **Purpose:** Encrypts a string using XOR encryption and encodes it in Base64.
  + **Parameters:**
    - text (String): The text to be encrypted.
    - key (String): The key used for XOR encryption.
  + **Returns:**
    - (String): The Base64 encoded, XOR encrypted text.
  + **Implementation:**
    - Applies XOR encryption on each character of the text using the key.
    - Encodes the encrypted text in Base64.

**Example Usage**

The lol function can be tested by passing an encoded string and verifying the decoded number:

const number = 12345;

const encodedString = encodeNumberToString(number); // Uncomment if encoding function is active

const decodedNumber = lol(encodedString);

console.log("Decoded Number:", decodedNumber);

**Control.js**

**WebRPD Control Panel Documentation**

**Overview**

The addVisibilityAndTransparencyControls function is a part of the WebRPD (Web-based Robotic Prosthetic Design) system, which provides a graphical user interface (GUI) for controlling various properties of 3D mesh objects in a scene. This function allows users to adjust visibility and transparency, manage materials, and interact with different jaw types and surface features through an interactive control panel.

**Features and Functions**

**1. Custom CSS Styling**

* **Scrollbar Customization**:
  + Adjusted scrollbar width and color for a consistent look and feel.
  + Scrollbars are hidden by default and only shown when hovered over.
* **Folder Styling**:
  + Folders in the GUI are styled with a black background and white text.
  + Close controls in folders are hidden for a cleaner appearance.
* **Icon Styling**:
  + Icons in the folder titles are enlarged and centered for better visibility.

**2. GUI Container Setup**

* **Positioning and Styling**:
  + The GUI container is positioned absolutely with a specified distance from the top and left.
  + The container has a fixed width and maximum height, with a custom scrollbar.
* **Black Bar Background**:
  + A black bar is created behind the GUI container to enhance visual integration and prevent layout issues.
  + The black bar adjusts its width and height based on the viewport size and whether the device is mobile.

**3. GUI Creation and Controls**

* **GUI Initialization**:
  + An instance of dat.GUI is created, and its DOM element is appended to the custom GUI container.
* **Visibility Controls**:
  + For each mesh child in the parentObject, a folder is created with a title icon based on the mesh name.
  + Each folder contains checkboxes to toggle the visibility of the corresponding mesh.
* **Transparency Controls**:
  + Sliders are added to adjust the transparency of the materials of each mesh.
* **Material Management**:
  + For non-surface meshes, additional controls are provided to switch between different material types (e.g., undercut and occlusion).
  + Surface meshes have controls to toggle between different surface properties (e.g., metallic).
* **Global Visibility Toggle**:
  + A button is added to the GUI to toggle the visibility of all meshes at once.

**4. Device-Specific Adjustments**

* **Mobile Device Adjustments**:
  + The scale and position of the GUI container are adjusted for better usability on mobile devices.

**5. External Control Button**

* **Toggle Button**:
  + An external button is created to open and close the GUI controls.
  + The button's appearance is customized for a professional look, with hover effects for better user interaction.

**6. Event Handling**

* **Visibility and Transparency Changes**:
  + Event listeners are added to checkboxes and sliders to dynamically update mesh properties based on user input.
* **Toggle Button Click Event**:
  + The button toggles the display of the GUI container and the black bar.

**7. Utility Functions**

* **isMobileDevice Function**:
  + Determines if the current device is mobile based on the user agent string.

**Code Usage**

To integrate this function into your project:

1. **Import the Function**:

import { addVisibilityAndTransparencyControls } from './control.js';

1. **Call the Function**:

addVisibilityAndTransparencyControls(parentObject, name, materialArray, jawType);

* + parentObject: The 3D object containing mesh children to be controlled.
  + name: Name identifier for the control panel.
  + materialArray: Array of materials used for different mesh configurations.
  + jawType: Configuration for different jaw types.

**ResetButton.js**

**WebRPD: ResetButton.js**

**Overview**

The resetbutton.js file is responsible for adding interactive UI elements to the WebRPD application, including a reset button, a rotation lock button, and a legend container. These elements enhance user experience by allowing users to reset the camera view, lock or unlock rotation, and view a legend with color-coded information.

**Key Features**

**1. Reset Button**

* **Functionality**: Allows users to reset the camera view and its properties.
* **UI Elements**:
  + **Icon**: An image (reset.png) representing the reset action.
  + **Text**: A label reading "Reset".
* **Behavior**:
  + When clicked, the button:
    - Adds a 'clicked' class to trigger a rotation animation on the icon.
    - Resets the camera's view to a pre-defined state by copying properties from a clone object.
    - Resets camera zoom and projection matrix.
    - Resets control target to the origin (0, 0, 0) and updates controls.
    - Removes the 'clicked' class after a short delay to reset the animation.

**2. Lock Rotation Button**

* **Functionality**: Toggles the rotation lock state, controlling whether rotation is allowed.
* **UI Elements**:
  + **Icon**: An image that changes based on the lock state (lock.png or unlock.png).
  + **Text**: A label reading "Rotation".
* **Behavior**:
  + When clicked, toggles the rotation lock state between locked and unlocked.
  + Updates the button’s image to reflect the current state.
  + Disables or enables rotation on the controls based on the lock state.
* **Auto-Click Feature**: Automatically clicks the button twice upon initialization for an undisclosed reason (possibly a feature to initialize the rotation state).

**3. Legend Container**

* **Functionality**: Displays a legend with color-coded information relevant to the application.
* **UI Elements**:
  + **Sections**: Each section displays a title and color-coded labels.
    - **Example Sections**:
      * **Undercut (mm)**: Displays ranges of values with corresponding colors.
      * **Occlusion (mm)**: Shows occlusion ranges with specific colors.
* **Behavior**:
  + Each section contains colored boxes and corresponding text to represent various values.
  + Responsive design adjustments based on screen width:
    - **1200px**: Increases container width to 35%.
    - **768px**: Adjusts width to 50% and modifies bottom position.
    - **480px**: Further adjusts width to 70%, reduces font sizes, and changes layout to fit smaller screens.

**Styles and Animations**

* **Reset Button Animation**: Utilizes CSS keyframe animations to rotate the reset icon when the button is clicked.
* **Responsive Design**: Custom CSS ensures that the layout and font sizes of the legend container adapt to different screen sizes for better usability.

**Implementation Details**

1. **CSS Styles**:

* Added styles directly in the document’s head for both the reset button and lock rotation button, including hover effects and animations.
* Custom CSS for responsive design ensures that the UI adjusts gracefully to various screen sizes.

1. **HTML Elements**:

* Created and appended buttons and legend containers dynamically to the document body.

1. **Event Listeners**:

* Attached click event listeners to handle interactions with the reset and lock rotation buttons.

1. **Dynamic Updates**:

* Functions like updateLockRotationButtonImage dynamically update button visuals based on the current state.