I'm not robot

Dr jennifer daniels turpentine protocol

getOndemandOrDownloadUrl(url) { return SP.Utils.getOndemandOrDownloadUrl(url); } persist(val, key) { const userId = null || 'guest'; return Alpine.$persist(val).as(SP.Utils.getLocalStorageKey(userId, key)); } fetchVariant(src) { return fetch(src, { method: 'HEAD' }) .then((response) => { Alpine.store('audio').variant = SP.Utils.extractVariant(response.url); }) .catch(() => { Alpine.store('audio').variant = null; }); } init() { const $audioRef = this.$refs.audio; const getOndemandOrDownloadUrl = this.getOndemandOrDownloadUrl; Alpine.store('audio', { episodeQueue: this.persist([], 'audio.episode_queue'), activeEpisode: this.persist(null, 'audio.active_episode'), playback_token: this.persist(null, 'audio.playback_token'), episodesProgress: this.persist({}, 'audio.episodes_progress'), episodesDuration: this.persist({}, 'audio.episodes_duration'), episodesCurrent: this.persist({}, 'audio.episodes_current'), episodesCompleted: this.persist({}, 'audio.episodes_completed'), playbackRate: 1, playing: false, loading: false, variant: null, }); } play() { if (this.playing) return; if (this.activeEpisode) { this._updateAudioSrc(this.activeEpisode); this._updateMediaSession(this.activeEpisode); this._trackAmplitudeEvent('played', this.activeEpisode); this._trackBackendPlaybackEvent('play', this.activeEpisode, this.getCurrent()); this.changePlaybackRate(this.playbackRate); $audioRef.currentTime = this.getCurrent(); $audioRef.play(); return; } else if (this.episodeQueue.length === 0) return; else { this.activeEpisode = this.episodeQueue[0]; this._dequeue(this.activeEpisode); this._updateAudioSrc(this.activeEpisode); this._updateMediaSession(this.activeEpisode); this._trackAmplitudeEvent('played', this.activeEpisode); this._trackBackendPlaybackEvent('play', this.activeEpisode, this.getCurrent()); $audioRef.play(); } } pause() { this._trackAmplitudeEvent('paused', this.activeEpisode); this._trackBackendPlaybackEvent('pause', this.activeEpisode, this.getCurrent()); $audioRef.pause(); } plus30() { if (!this.activeEpisode) return; this.setCurrentTime(this.getCurrent() + 30); this._trackAmplitudeEvent('seek_forward', this.activeEpisode); var mediaController = { activeEpisode: null, episodeQueue: [], // Play a new episode playNow: function (episode) { if (this._equals(this.activeEpisode, episode)) { return this.play(); } this._stop(); this._enqueueHead(episode); this.play(); this._emitEpisodePlayEvent(); }, // Toggle playback of current episode playPause: function (episode) { if (this._equals(this.activeEpisode, episode)) { if (!this.loading) { return this.playing ? this.pause() : this.play(); } } else { return this.playNow(episode); } }, // Seek within current episode seek: function (progress) { const duration = $audioRef.duration || this.activeEpisode.duration / 1000; const current = duration * progress; this.setCurrentTime(current); this._trackAmplitudeEvent('seek_at', this.activeEpisode); this._trackBackendPlaybackEvent('seek', this.activeEpisode, this.getCurrent()); return this.play(); }, // Move to next episode next: function (automatic) { this._trackAmplitudeEvent('next', this.activeEpisode, { automatic }); if (automatic) { this.setProgress(0); this.setDuration(null); this.setCurrent(0); } return this._stop().play(); }, // Add episode to queue addToQueue: function (episode) { if (!this._equals(this.activeEpisode, episode)) { this._dequeue(episode); this.episodeQueue = [...this.episodeQueue, episode]; if (this.episodeQueue.length === 1 && !this.activeEpisode) { this.activeEpisode = this.episodeQueue[0]; this._dequeue(this.activeEpisode); } } return this._trackAmplitudeEvent('queue_added', episode, { queue_length: this.countLoadedEpisodes() }); }, // Remove episode from queue removeFromQueue: function (episode) { this._dequeue(episode); return this._trackAmplitudeEvent('queue_removed', episode, { queue_length: this.countLoadedEpisodes() }); }, // Update playback rate changePlaybackRate: function (rate) { this.playbackRate = rate; $audioRef.playbackRate = rate; } }; this.updateActiveEpisodeTime = (time) => { setCurrent(time); $audioRef.currentTime = time; }; getProgress = () => { if (!activeEpisode) return 0; return episodesProgress[activeEpisode.episode_id]; }; getEpisodeProgress = (episode) => episodesProgress[episode.episode_id] || 0; getEpisodeCurrent = (episode) => episodesCurrent[episode.episode_id] || 0; getEpisodeDuration = (episode) => { const durationFromEpisodes = episodesDuration[episode.episode_id] || episode.duration / 1000; return durationFromEpisodes; }; getEpisodeCompleted = (episode) => episodesCompleted[episode.episode_id] || false; getEpisodeMinLeft = (episode) => { const current = getEpisodeCurrent(episode); const duration = getEpisodeDuration(episode); return SP.Utils.formatTimeLeft(duration - current, SP.Viewer.language); }; setProgress = (progress) => { if (!activeEpisode) return; episodesProgress[activeEpisode.episode_id] = progress; }; getDuration = () => { if (!activeEpisode) return null; const durationFromEpisodes = episodesDuration[activeEpisode.episode_id] || activeEpisode.duration / 1000; return durationFromEpisodes; }; setDuration = (duration) => { if (!activeEpisode) return; episodesDuration[activeEpisode.episode_id] = duration; }; getCurrent = () => episodesCurrent[activeEpisode.episode_id] || 0; setCurrent = (current) => { if (!activeEpisode) return; episodesCurrent[activeEpisode.episode_id] = current; }; getCompleted = () => episodesCompleted[activeEpisode.episode_id] || false; setCompleted = (completed) => { if (!activeEpisode) return; episodesCompleted[activeEpisode.episode_id] = completed; }; updateEpisode = (episode) => { // Update active episode if needed if (_isEpisodeActive(episode)) { activeEpisode = episode; } // Update episode in queue const updatedQueue = episodeQueue.map((queuedEpisode) => { if (_equals(queuedEpisode, episode)) return episode; return queuedEpisode; }); // Update progress, duration and current time for non-active episodes if (!_isEpisodeActive(episode) && episode.progress) { const newProgress = (episode.progress / episode.duration) * 100; const newDuration = episode.duration / 1000; const newCurrent = episode.progress / 1000; const newCompleted = episode.completed_at !== undefined || episode.completed_at !== null; episodesProgress[episode.episode_id] = newProgress; episodesDuration[episode.episode_id] = newDuration; episodesCurrent[episode.episode_id] = newCurrent; episodesCompleted[episode.episode_id] = newCompleted; } }; getEpisodePlayButtonState = (episode) => { if (shouldDisplayLoading(episode)) return 'LOADING'; if (shouldDisplayPause(episode)) return 'PAUSE'; if (shouldDisplayPlay(episode) && getEpisodeCompleted(episode) || !getEpisodeCurrent(episode)) return 'PLAY'; return 'RESUME'; }; displayEpisodeProgress = (episode) => { if (isEpisodePlaying(episode)) return true; if (!getEpisodeCompleted(episode) && getEpisodeCurrent(episode)) return true; return false; }; displayEpisodeDuration = (episode) => { if (isEpisodePlaying(episode)) return false; if (!getEpisodeCompleted(episode) && getEpisodeCurrent(episode)) return false; return true; }; resetProgress = () => { setProgress(0); setCompleted(false); }; Here is a paraphrased version of the given text: The session API is used to display current episode information on the device's lock screen. The implementation is limited and has some bugs, so this should be considered a best effort. The code defines several functions, including _updateMediaSession() which updates the media session metadata and sets action handlers for seeking, playing, pausing, and skipping tracks. The code also includes event handling for episode play, audio loading, time updating, and error handling. This JavaScript code snippet defines an Alpine.js component for handling playback tokens and tracking backend playback progress. The `fetchAndStorePlaybackToken` method fetches a playback token from the backend and stores it in the Alpine store, while the `ensurePlaybackToken` method ensures that a valid playback token is present before refreshing it if necessary. The code also defines private methods `_trackBackendPlaybackEvent` and `_trackBackendPlaybackProgress`, which track backend playback events and progress, respectively. The component uses intervals to periodically check for expired playback tokens and refresh them as needed. In this article, we explore how Dr. Jennifer Daniels, a renowned alternative healing physician, is helping people heal from diseases since 1985. She holds a BA degree with honors from Harvard University, an MD from the University of Pennsylvania, and an MBA. I'm joined by Dr. Daniels, an expert in Healthcare Administration from Wharton, and author of the acclaimed book "Do You Have the Guts to Be Beautiful?". We dive into her fascinating journey, from studying medicine to becoming a doctor in her community, and discovering the potential healing properties of pure gum spirits of turpentine - which ultimately led her to flee the US for Panama. We explore issues plaguing science and healthcare, including germ theory, disease causes, and recovery methods. I had the pleasure of speaking with Dr. Daniels, who shares her insights on the importance of pure gum spirits in healing. You can learn more about Dr. Daniels' work at Vitality Cycles (and stay updated on Humanley's Telegram channel (. Please note that this podcast is for general information purposes only, does not constitute health advice, and shouldn't replace consulting a primary healthcare practitioner.