# Synchronization: A Story

**Kevin Putrajaya**

Once upon a time, when building a *web-app*, I faced these challenges:

- **Multi-device sync** across tabs and devices
- **Real-time updates** without page refreshes
- **Seamless connect and disconnect** at anytime
- **Lightweight client** implementation

... Alright, I was building **board game apps**.

# Alternatives

## ⛔ Browser APIs — *local and same-browser only*

- `BroadcastChannel`: Communication without persistence
- `SharedWorker`: Complex custom logic and processing
- `localStorage` **events**: Triggered for every value change

## ⚠️ SaaS — *vendor lock-in and more bloated*

- Pusher: Free for 200k daily messages
- Firebase RTDB: Free for 1GB storage & 10GB monthly bandwidth

# Alternatives (cont'd)

✅ **Custom solution — *why easy if hard can?***

- **WebSocket**: Lightweight but low-level
- **Socket.IO**: Abstracted with polling fallback

# Solution

## 🗄️ Server-side

- **WebSocket**: Native real-time communication
- **Cache backend**: In-memory storage with TTL
- **Room-based**: Message routing based on keys

## 💻 Client-side

- **Lightweight SDK**: 1kB download size
- **Easy implementation**: One statement setup
- **URL param trigger**: Integrate without UI changes

# Sample Integration

## ⌨️ Implement

```
new PubSub({
  host: 'ws://localhost:8075',
  appKey: 'sample',
  getData: () => data,
  setData: (v) => (data = v),
});
```

## ▶️ Activate

```
<url>?k=<key>
```

# Walkthrough and Demo

# Conclusion

- **Solved a practical need**: Enabled synchronization across devices
- **Maintained a lean approach**: Minimized complexity wherever possible
- **Prioritized client simplicity**: Designed with client-focused requirements

> **Those who build their own WebSocket server do not have vendor lock-in problem.**
>
> — Confucius