PSP Repository Health Check: Missing Critical Components

The PSP-PersistentSessionsProtocol appears to be in early development stages with several critical components missing from its structure. While the foundational files exist, implementation code and supporting infrastructure require significant development to meet the protocol's ambitious cross-framework persistence goals.

Bottom line up front

PSP-PersistentSessionsProtocol has essential configuration files in place but lacks actual implementation code in src/ and packages/ directories Escape DAST and requires comprehensive documentation, testing infrastructure, and examples. Cursa The repository needs substantial development across all four architectural layers (Capture, Serialization, Storage, Replay) to fulfill its cross-framework persistence protocol goals. Immediate focus should be on developing core implementation files, creating framework adapters, and establishing proper documentation. GitHub GitHub

Current repository structure

The repository currently contains these basic files:

- README.md
- package.json
- LICENSE
- CONTRIBUTING.md
- tsconfig.json
- CLAUDE.md

Based on this structure and standard TypeScript project organization, GitHub the likely current directory structure looks like this:

```
/PSP-PersistentSessionsProtocol
├── README.md              # Project overview and documentation
├── package.json           # Project metadata and dependencies
├── LICENSE                # License information
├── CONTRIBUTING.md        # Contribution guidelines
├── tsconfig.json          # TypeScript configuration
└── CLAUDE.md              # AI assistant documentation
```

This indicates the repository is in a very early stage with configuration files but potentially missing critical implementation code.

Missing critical components

Implementation code

The repository appears to be missing the core implementation code that would be expected in the `src/` directory. GitHub This should include:

- **Core protocol definition files** defining interfaces for all layers
- **Session Capture Layer implementation** for intercepting browser events Decipher AI
- **Serialization Layer implementation** for converting captured data to storable format GitHub
- **Storage Layer implementation** with adapters for different storage mechanisms GitHub
- **Replay Layer implementation** for session reconstruction and playback DEV Community +5

Framework adapters

A cross-framework protocol would typically have framework-specific implementations in the `packages/` directory, Basedash which appear to be missing:

- **React integration** package with hooks and components
- **Vue integration** package with composables and components
- **Angular integration** package with services and directives
- **Vanilla JS reference** implementation GitHub +4

Documentation

While README.md exists, a comprehensive protocol should have extensive documentation in a `docs/` directory: Cursa

- **API reference** documentation for each layer
- **Architecture diagrams** explaining the protocol design
- **Integration guides** for different frameworks
- **Usage examples** showing implementation patterns Make a README +2

Testing infrastructure

The repository lacks evidence of testing infrastructure: Sean Coughlin's Blog

- **Unit tests** for each protocol layer
- **Integration tests** across layers
- **End-to-end tests** for full protocol functionality
- **Test configuration files** GitHub +2

Example implementations

Working examples demonstrating the protocol are critical but missing: GitHub

- **Basic usage examples** showing core functionality GitHub
- **Framework-specific examples** for React, Vue, Angular, etc.
- **Custom configuration examples** demonstrating extension points GitHub GitHub

Ideal directory structure

Based on TypeScript best practices and similar protocol implementations, Cursa a complete PSP repository should include:

```
/PSP-PersistentSessionsProtocol
├── README.md
├── package.json
├── LICENSE
├── CONTRIBUTING.md
├── tsconfig.json
├── CLAUDE.md
├── src/
│   ├── index.ts                # Main entry point
│   ├── types.ts                # Core type definitions
│   ├── capture/                # Session Capture Layer
│   │   ├── index.ts
│   │   ├── dom-events.ts
│   │   ├── state-tracker.ts
│   │   └── filters.ts
│   ├── serialization/          # Serialization Layer
│   │   ├── index.ts
│   │   ├── serializers.ts
│   │   ├── compression.ts
│   │   └── schemas.ts
│   ├── storage/                # Storage Layer
│   │   ├── index.ts
│   │   ├── adapters/
│   │   │   ├── local-storage.ts
│   │   │   ├── indexed-db.ts
│   │   │   └── custom.ts
│   │   ├── encryption.ts
│   │   └── migration.ts
│   └── replay/                 # Replay Layer
│       ├── index.ts
│       ├── player.ts
│       ├── controls.ts
│       └── renderer.ts
├── packages/                   # Framework implementations
│   ├── react/
│   ├── vue/
│   ├── angular/
│   └── vanilla/
├── test/
│   ├── unit/
│   ├── integration/
│   └── e2e/
├── docs/
│   ├── api/
│   ├── guides/
│   ├── examples/
│   └── architecture/
└── examples/
    ├── vanilla/
    ├── react-app/
    ├── vue-app/
    └── angular-app/
```

Stack Overflow +4

Evaluation against architectural requirements

The repository appears to be missing implementation of all four critical architectural layers:

1. **Session Capture Layer**: No evidence of event listener implementation, DOM mutation tracking, or framework-specific capture mechanisms Datadog Tealeaf
2. **Serialization Layer**: No implementation for transforming captured events into storable formats DEV Community riptide
3. **Storage Layer**: No storage adapters or persistence mechanisms implemented MDN Web Docs Stack Overflow
4. **Replay Layer**: No playback engine or session reconstruction implementation Fullstory +7

Recommendations for next steps

Based on this analysis, here are the recommended next steps to complete the repository:

1. Core implementation development (highest priority)

- Create the basic directory structure for src/ following the layered architecture Stack Overflow
- Implement core interfaces for each protocol layer Decipher AI
- Develop the Session Capture Layer with DOM event listeners Restack Rrweb
- Build the Serialization Layer with JSON transformation GitHub
- Implement a basic Storage Layer with localStorage adapter Webdevtutor
- Create a minimal Replay Layer for session reconstruction Qualtrics +8

2. Framework integration development

- Set up the packages/ directory structure Basedash
- Implement React bindings with hooks and components Webdevsimplified
- Create Vue integration with composables Restack
- Develop Angular services and directives Restack
- Build a framework-agnostic reference implementation Stack Overflow +6

3. Documentation expansion

- Create detailed API documentation for each layer ChainSafe
- Develop architectural diagrams showing layer interaction
- Write usage guides for different implementation scenarios GitHub
- Provide framework-specific integration documentation Visualstudio +5

4. Testing infrastructure establishment

- Implement unit tests for each layer component Sean Coughlin's Blog +2
- Create integration tests across layers Sean Coughlin's Blog +2
- Develop end-to-end tests for complete protocol functionality Sean Coughlin's Blog Raygun Blog
- Set up continuous integration workflows Procodebase +7

5. Example implementation creation

- Build working examples for each supported framework GitHub
- Create reference implementations showing different configuration options GitHub
- Develop demos showcasing the protocol capabilities Stack Overflow +5

Conclusion

The PSP-PersistentSessionsProtocol repository has the basic foundational files but lacks the critical implementation components needed for a functional cross-framework session persistence protocol. The repository needs substantial development work across all four architectural layers to fulfill its goals.

Focus should first be on implementing the core functionality in the src/ directory, following the layered architecture design, before expanding to framework-specific packages and comprehensive documentation. By following the recommended steps, the repository can evolve from its current skeletal state into a robust, production-ready protocol implementation. Design Patterns +2