BidNow - Sprint Review Document

A real-time auction platform where registered users can bid on listed items, track ongoing auctions, and manage their listings. Admins can manage users, listings, and track revenue metrics.

Sprint Name: Sprint 2 – Interactivity, SPA Architecture, API Consumption
Sprint Goal: Enable SPA structure and dynamic data handling.
Duration: 16 hours.
Team Project: BidNow – Online Auction System
Pod Members:

- Geethu Joseph
- Govindh RM
- Jayashree VS
- Karthika RS
- Kushagra Srivastava
- Mayukh Paul.

Submission Date: 19 June, 2025

❖ What Was Planned vs. What Was Delivered

- **User Stories for each member task:**

Member 1 (Geethu Joseph): Convert UI logic to TypeScript and define interfaces (Auction, User, Bid).

- o As a team member, I want to define interfaces like Auction, User, and Bid so that data models are consistent across the project.
- o As a developer, I want to convert existing JS logic into TypeScript so that the code becomes type-safe and more maintainable.
- o As a collaborator, I want to share reusable interface files so that other members can maintain consistency when handling shared data types.

Member 2 (Govindh RM): Fetch mock auction data from JSON and render listings.

- o As a user, I want to see auction listings dynamically rendered from JSON data so that I can view real-time-like listings.
- o As a developer, I want to use fetch() to load mock data and display it dynamically using DOM manipulation.
- o As a UI designer, I want the auction data to load with a loading spinner or message, so users understand content is being fetched.

Member 3 (Jayashree VS): Implement dynamic search and category filtering.

- o As a bidder, I want to filter auctions by categories so that I can easily find items I'm interested in.
- o As a user, I want a search bar that dynamically filters items by title or keyword as I type.
- o As a user, I want the search and category filters to work together so that I can narrow down results efficiently.

Member 4 (Karthika RS): Add routing for pages: Auctions, My Bids, Dashboard, Contact.

- o As a user, I want to navigate between sections (like Auctions, Dashboard, My Bids) without reloading the page so that my experience is seamless.
- o As a developer, I want to use SPA routing so that page state is managed via client-side logic.
- o As a registered user, I want a dashboard where I can view and review the items. I've placed bids on, so I can track my bidding activity and plan accordingly.

Member 5 (Kushagra Srivastava): Enable lazy loading for auction detail pages.

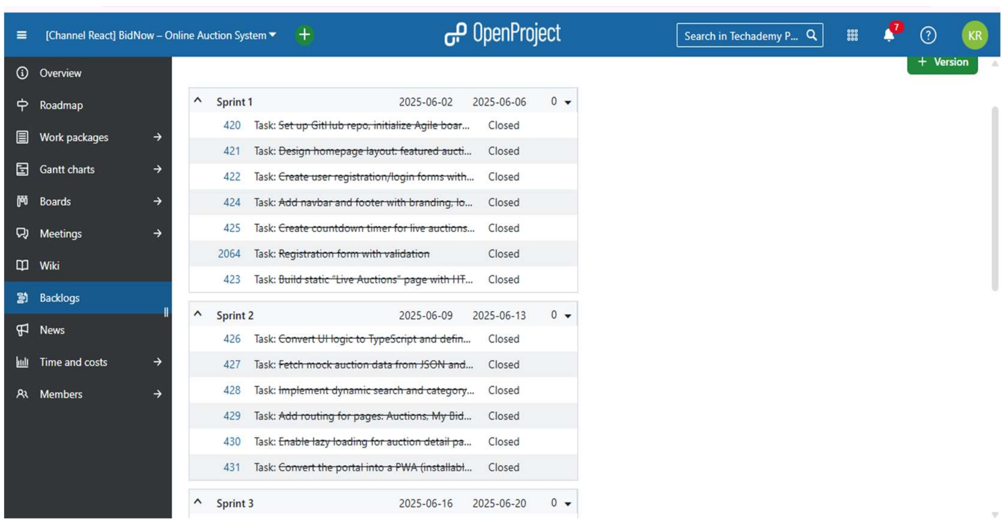
- o As a user, I want auction details to load only when I click them so that the initial page loads faster.
- o As a developer, I want to implement modular loading logic for better performance
- o As a developer, I want to split the auction detail component into a separate module so that it only loads when needed, reducing initial load time.

Member 6 (Mayukh Paul): Convert the portal into a PWA (installable + offline support).

- o As a user, I want to install the web app on my device like a native app so that it feels more integrated.
- o As a developer, I want to enable service workers to cache essential assets for offline use.
- o As a user with slow internet, I want the app to load from cache instantly so I can use core features even if I'm offline.

Planned Task	**Status**	**Delivered**
Convert core JS logic to TypeScript with interfaces	Completed	All major functions converted, interfaces defined in types.ts
Load and render auctions using JSON	Completed	Used mock JSON file and fetch() to dynamically populate auction cards
Implement live search and category-based filtering	Completed	JavaScript search/filter applied to all auction listings
Add SPA routing for pages: Auctions, My Bids, Dashboard, Contact.	Completed	Used client-side router to switch pages between My Bids, Dashboard, Contact
Lazy load auction details	Completed	Details only load on click using dynamic imports.
Implement PWA support	Completed	Manifest.json and service worker added; app installable and functional offline

❖ **OpenProject sprint status screenshot:**



Sprint 1		2025-06-02	2025-06-06	0
420	Task: ~~Set up GitHub repo, initialize Agile boar...~~	Closed		
421	Task: ~~Design homepage layout: featured aucti...~~	Closed		
422	Task: ~~Create user registration/login forms with...~~	Closed		
424	Task: ~~Add navbar and footer with branding, lo...~~	Closed		
425	Task: ~~Create countdown timer for live auctions...~~	Closed		
2064	Task: ~~Registration form with validation~~	Closed		
423	Task: ~~Build static "Live Auctions" page with HT...~~	Closed		
Sprint 2		2025-06-09	2025-06-13	0
426	Task: ~~Convert UI logic to TypeScript and defin...~~	Closed		
427	Task: ~~Fetch mock auction data from JSON and...~~	Closed		
428	Task: ~~Implement dynamic search and category...~~	Closed		
429	Task: ~~Add routing for pages: Auctions, My Bid...~~	Closed		
430	Task: ~~Enable lazy loading for auction detail pa...~~	Closed		
431	Task: ~~Convert the portal into a PWA (installabl...~~	Closed		
Sprint 3		2025-06-16	2025-06-20	0

❖ **Challenges Faced**

1. **TypeScript Compilation Issues (Member 1)**
   - JavaScript logic had implicit any types, leading to multiple TypeScript compile-time errors.
   - Additionally, many existing pages had inline script logic and lacked modular structure, making it difficult to isolate logic into reusable TypeScript modules. Converting all pages to TS was a major challenge, requiring careful refactoring and repeated debugging.
2. **JSON Fetch CORS Simulation (Member 2)**
   - Some browsers blocked access to local files entirely, requiring the use of a dev server like live-server.
   - This setup added additional configuration time and debugging, especially when switching between environments.
3. **Search Filtering Conflicts (Member 3)**
   - Applying search and category filter simultaneously cleared previous results unintentionally.
   - Updating the filtered results array triggered repeated re-rendering, slowing down performance.
4. **Routing Not Working on Refresh (Member 4)**
   - Refreshing or directly entering a route URL (e.g., #dashboard) didn't load the corresponding content.
   - Most critically, only Contact and My Bids pages worked reliably with SPA routing. Other pages like Auctions and Dashboard were too large or complex (due to heavy inline JavaScript and deeply nested HTML) to render dynamically via JavaScript routing without major performance or rendering issues. This limited the true potential of routing in this sprint.

5. **Lazy Loading Timing (Member 5)**
   - Components meant to be lazy-loaded were accessed before they were fully fetched or rendered.
   - Event handlers were not binding correctly because the target DOM was not yet available during initial load.
6. **Service Worker Caching Bugs (Member 6)**
   - Old JS/CSS files were served from cache even after updates, causing layout or logic bugs.
   - Changes to files didn't reflect until hard refresh; fixed by updating cache version names and deleting stale caches.

❖ **Learnings**

1. **TypeScript Integration:**
   - Improved type safety by defining clear interfaces, which reduced bugs caused by inconsistent data structures.
   - Migrating JS to TS encouraged better organization and early detection of logic errors during development.
2. **Modular Design:**
   - Created reusable components (e.g., auction cards, filters) that reduced code duplication across pages.
3. **Search & Filter Logic:**
   - Combined real-time keyword search with category filtering to enable more refined and responsive results.
   - Used event listeners and filter() logic to update the UI without full page reloads.
4. **Routing Architecture:**
   - Implemented route-based content rendering with JavaScript, enabling smooth transitions between views.
5. **PWA Setup:**
   - Registered service workers to cache key assets like HTML, CSS, and JS files for offline access.
   - Created a manifest.json to make the app installable on mobile and desktop, mimicking native app behaviour.

❖ **Deliverables Summary**

1. TypeScript interfaces (Auction, User, Bid) defined in types.ts for consistent and type-safe data handling.

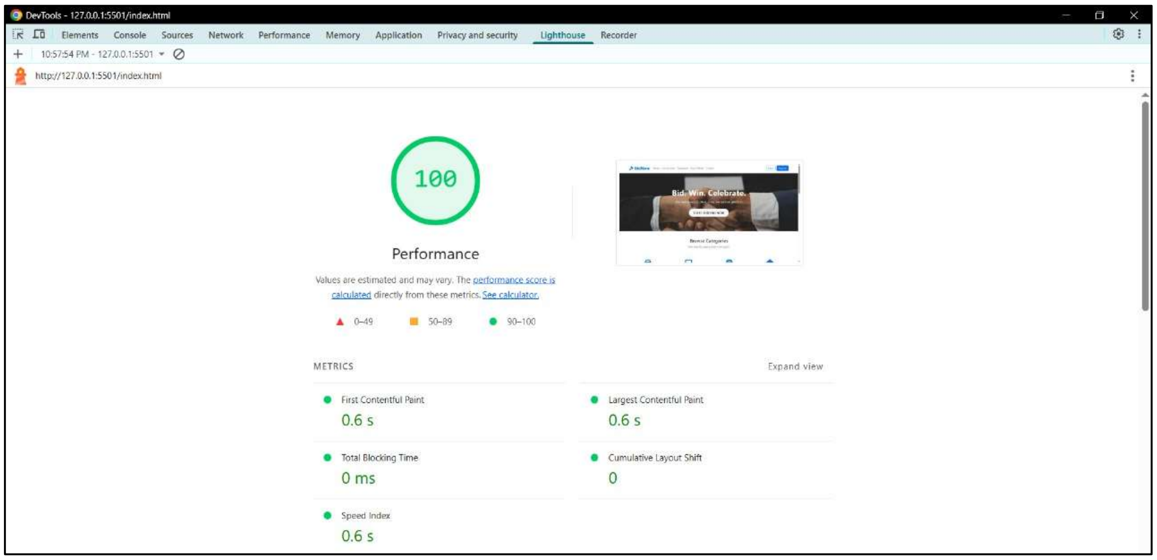
2. Fully functional dynamic search bar and category filter integrated with real-time updates.

3. Auction data fetched from data/auctions.json and dynamically rendered using TypeScript-based DOM manipulation.

4. Routing enabled for #auctions, #my-bids, #dashboard, and #contact, allowing navigation without full page reloads.

5. Auction detail pages lazy-loaded using dynamic import logic to improve initial load performance.

6. PWA setup completed: manifest.json added for installable experience, and service-worker.js implemented for offline support and asset caching.



100

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS — Expand view

● First Contentful Paint
0.6 s

● Largest Contentful Paint
0.6 s

● Total Blocking Time
0 ms

● Cumulative Layout Shift
0

● Speed Index
0.6 s

Fig. Lighthouse Audit for website after implementing PWA

Conclusion:
Sprint 2 successfully enhanced the BidNow platform by introducing interactivity, dynamic content rendering, and offline accessibility. All planned tasks were executed collaboratively, with continued adherence to Agile practices and version control. The team gained practical experience with TypeScript, JSON handling, modular design, and routing configuration. With dynamic data loading, seamless page navigation, and installable app support now in place, the platform is well-prepared for future sprints focusing on backend integration, user authentication, and real-time bidding functionality.