OSVERSE: INTERACTIVE OPERATING SYSTEMS SCHEDULING VISUALIZER AND AR DEMONSTRATOR

Project Report

Department of Computer Science
CHRIST (Deemed to be University)

Project Repository: https://github.com/shreyjain14/osverse-final
Live Website: https://osverse.shreyjain.me

Submitted by
Shrey Jain, Sagar Sharma, and Jerin K Joseph

Academic Year: 2025

August 20, 2025

CERTIFICATE

This is to certify that the project report titled **OSVerse: Interactive Operating Systems Scheduling Visualizer and AR Demonstrator** is a bonafide record of the work carried out by the student team under the guidance of the project supervisor, submitted to the CHRIST (Deemed to be University), in partial fulfillment of the requirements for the award of the degree.

Guide Signature	**Head of Department**
Date:	Date:

Place: Bengaluru

ACKNOWLEDGMENTS

We express our sincere gratitude to our guide and mentors for their continuous support and guidance. We also thank all contributors and the open-source community whose libraries and frameworks enabled the development of OSVerse. Finally, we acknowledge our peers and family members for their encouragement throughout this work.

ABSTRACT

The OSVerse project is an interactive, web-based visualization platform for classical CPU scheduling algorithms augmented with immersive Augmented Reality (AR) experiences. Built with Next.js, TypeScript, and Tailwind CSS, OSVerse provides animated Gantt charts, queue visualizations, and AR model viewers to enhance conceptual understanding of operating systems scheduling. The platform implements algorithms such as FCFS, SJF (preemptive and non-preemptive), Priority (preemptive and non-preemptive), Round Robin, HRRN, LJF, Lottery, Fair Share, EDF, Multilevel Queue, and Multilevel Feedback Queue. Major outcomes include an extensible animation framework, consistent UI components, and an API-ready model-view layer. The system is deployed at https://osverse.shreyjain.me with source code at https://github.com/shreyjain14/osverse-final. Recommendations include expanding test coverage, adding accessibility audits, and integrating backend persistence for user scenarios.

Contents

LIST OF TABLES

List of Tables

List of Figures

LIST OF ABBREVIATIONS

AR Augmented Reality

CPU Central Processing Unit

EDF Earliest Deadline First

FCFS First-Come, First-Served

HRRN Highest Response Ratio Next

LJF Longest Job First

MLFQ Multilevel Feedback Queue

MLQ Multilevel Queue

RR Round Robin

SJF Shortest Job First

UI User Interface

UX User Experience

WebXR Web Extended Reality APIs

1. INTRODUCTION

This chapter introduces the OSVerse project, summarizes its motivation and scope, and outlines the structure of the report. OSVerse is a web-native, interactive visualization suite for Operating Systems CPU scheduling, paired with optional Augmented Reality (AR) experiences to deepen conceptual understanding through spatial representations of execution timelines.

BACKGROUND AND MOTIVATION

Operating Systems (OS) are the backbone of modern computing, responsible for managing hardware resources, providing essential services, and ensuring the smooth execution of applications. One of the most critical tasks performed by an OS is process scheduling, which determines the order and manner in which processes are allocated CPU time. The efficiency and fairness of scheduling algorithms directly impact system performance, user experience, and resource utilization.

The study of CPU scheduling is a foundational topic in computer science curricula. However, traditional teaching methods—such as static diagrams, textbook examples, and classroom lectures—often fail to convey the dynamic and interactive nature of scheduling decisions. Concepts like preemption, context switching, starvation, and fairness are best understood through visualization and experimentation. This gap in pedagogy motivated the creation of OSVerse: a platform designed to make scheduling algorithms tangible, interactive, and engaging for learners.

PROJECT VISION

The project is a collaborative effort by undergraduate students, guided by faculty and supported by the open-source community. OSVerse is built with Next.js, TypeScript, and Tailwind CSS, and is available as a live website and open-source repository.

STRUCTURE OF THE REPORT

This report is organized to provide a comprehensive overview of OSVerse, from its conception and design to implementation, testing, and future directions. Each chapter delves into specific aspects of the project, ensuring that readers gain both theoretical and practical insights into the development and impact of the platform.

1.1 PROJECT DESCRIPTION

CPU scheduling is central to operating systems. Classical pedagogy relies on static diagrams, chalkboard timelines, or slides. These approaches under-communicate dynamic behavior: preemption points, context switches, fairness over time, and interactions between arrivals, priorities, and quantum sizes. OSVerse addresses these limitations by offering:

- Interactive inputs for processes (arrival, burst, priority) and algorithm parameters (e.g., time quantum).
- Animated Gantt charts and queue visualizations that show execution and waiting states step-by-step.
- Algorithm coverage across FCFS, SJF (preemptive/non-preemptive), Priority (preemptive/non-preemptive), Round Robin, HRRN, LJF, Lottery, Fair Share, EDF, Multilevel Queue, and Multilevel Feedback Queue.
- Optional AR export of the execution timeline to a 3D model (.glb) for immersive viewing on compatible devices.

OSVerse is implemented with Next.js (React + TypeScript) and Tailwind CSS for the UI; Recharts and custom Canvas animations for charts; Framer Motion for micro-interactions; and a serverless API route that generates glTF/GLB 3D models for AR viewing. The live website is https://osverse.shreyjain.me and the source code is hosted at https://github.com/shreyjain14/o final.

1.2 EXISTING SYSTEM

Multiple teaching aids exist: printed examples in textbooks, slide decks, Java applets or desktop simulators, and assorted web demos. These typically focus on a small subset of algorithms (often FCFS, non-preemptive SJF, and simple Round Robin), and present limited animation fidelity. Notable gaps include:

- **Breadth of algorithms:** Advanced or less common strategies (e.g., HRRN, Lottery, Fair Share, EDF, MLFQ) are often missing.
- **Rich animations:** Many tools show static blocks rather than synchronized, stepwise executions with helpful overlays and legends.

- **Modern web stack:** Some older tools rely on outdated technologies (e.g., Java applets) or are not mobile-friendly.
- **AR augmentation:** Few, if any, integrate AR or 3D exports that can be explored spatially.
- **Accessibility:** Keyboard navigation, color contrast, and screen-reader friendly layouts are often overlooked.

OSVerse is designed to bridge these gaps with a modular, accessible, web-native platform that is easily deployable and extensible.

1.3 OBJECTIVES

Design and implement a web-native, extensible, and accessible platform that visualizes multiple CPU scheduling algorithms through synchronized animations and charts, and optionally exports AR-ready 3D timelines, to improve learner comprehension and engagement.

1.4 PURPOSE, SCOPE AND APPLICABILITY

1.4.1 Purpose

OSVerse aims to transform how CPU scheduling is taught and learned. By coupling precise algorithm implementations with dynamic visuals and optional AR representations, it helps learners form accurate mental models of execution, preemption, waiting time, and fairness.

1.4.2 Scope

The project focuses on process scheduling algorithms and their visualization. It implements a standard UI for process input, reusable animation components, a Gantt chart renderer, and an AR export API. Assumptions include single-CPU execution, discrete time units for clarity, and deterministic inputs. The methodology follows iterative development with component-first design, emphasizing correctness, clarity, and responsiveness.

1.4.3 Applicability

Applications include CS classrooms, lab demonstrations, flipped learning modules, technical talks, outreach events, and self-study. Educators can demonstrate phenomena like convoy effect, starvation mitigation, and quantum tuning. Learners can interactively vary inputs to observe outcomes immediately.

1.5 OVERVIEW OF THE REPORT

Chapter 2 analyzes the problem and requirements. Chapter 3 presents the architecture, modules, and interface design. Chapter 4 outlines the implementation and key code. Chapter 5 covers testing. Chapter 6 summarizes outcomes, limitations, and future work. Appendices include the repository structure, user manual, and planning artifacts.

2. SYSTEM ANALYSIS AND REQUIREMENTS

This chapter defines the problem, specifies requirements, and presents conceptual models of the system. It documents what the system must do without prescribing how it must be built.

2.1 PROBLEM DEFINITION

Learners need to understand both the *what* and the *why* of CPU scheduling decisions. Static content cannot capture preemption timing, queue dynamics, or fairness. The problem decomposes into several core challenges:

- **Algorithm engines:** Correctly compute schedules given arrivals, bursts, priorities, and quantum. The engines must handle edge cases such as simultaneous arrivals, zero burst times, and priority inversion. Future additions will include support for multi-core scheduling, distributed systems, and real-time constraints. See Scheduling Concepts.
- **Visualization:** Translate schedules into time-based animations and Gantt/queue views with legends. Visualizations should be customizable, allowing users to select color schemes, zoom levels, and annotation overlays. Planned features include 3D Gantt charts and timeline comparisons across algorithms. Explore Animated Gantt Demo.
- **Interaction:** Let users add/remove processes, adjust parameters, and replay animations. The platform will introduce scenario saving, batch editing, and collaborative input modes. Future work includes voice input, gesture controls, and integration with external data sources. Try Process Table.
- **AR export:** Convert timelines into a 3D model for immersive viewing. The AR export feature will be expanded to support real-time overlays, collaborative AR sessions, and integration with external AR platforms. Learn about AR Viewer.
- **Performance/accessibility:** Maintain responsiveness on typical devices and be usable by a broad audience. Accessibility audits, performance profiling, and compatibility testing are ongoing. Planned enhancements include adaptive layouts, localization, and alternative input methods. Read WCAG Guidelines.

In addition to these core challenges, OSVerse aims to support advanced research and experimentation. Users will be able to simulate large-scale scenarios, analyze performance metrics, and export results for further study. Planned integrations include statistical reporting, machine learning-based scheduling optimization, and support for external datasets.

2.2 REQUIREMENTS SPECIFICATION

Functional Requirements

1. Accept process inputs: arrival time, burst time, and priority (where applicable).
2. Compute schedules for FCFS, SJF (preemptive/non-preemptive), Priority (preemptive/non-preemptive), Round Robin, HRRN, LJF, Lottery, Fair Share, EDF, MLQ, and MLFQ.
3. Render animated Gantt charts and legends; provide play/pause, speed control, and reset.
4. Provide summary metrics: waiting time (WT), turnaround time (TAT), and their averages.
5. Export Gantt data to a downloadable GLB model for AR viewing.

Non-Functional Requirements

- **Correctness:** Visuals must accurately reflect algorithm logic.
- **Performance:** Smooth animations at typical frame rates on mainstream laptops/mobile devices.
- **Accessibility:** Keyboard navigation, sufficient color contrast, ARIA labels where practical.
- **Compatibility:** Support modern Chromium-based browsers, Safari, and Firefox; progressive degradation for limited WebXR.
- **Extensibility:** Modular components to add new algorithms and visualizations easily.

Table 2.1: Key Requirements Summary

oprule Category	Examples
Functional	Input processes, compute schedules, animate Gantt, export AR model
Non-Functional	Accessibility, cross-browser, responsive UI, smooth animations
Constraints	Browser camera permission, device WebXR support, single-CPU model

2.3 BLOCK DIAGRAM

Figure 2.1 depicts the high-level data flow from user inputs to algorithm engines, to visual renderers and AR export.

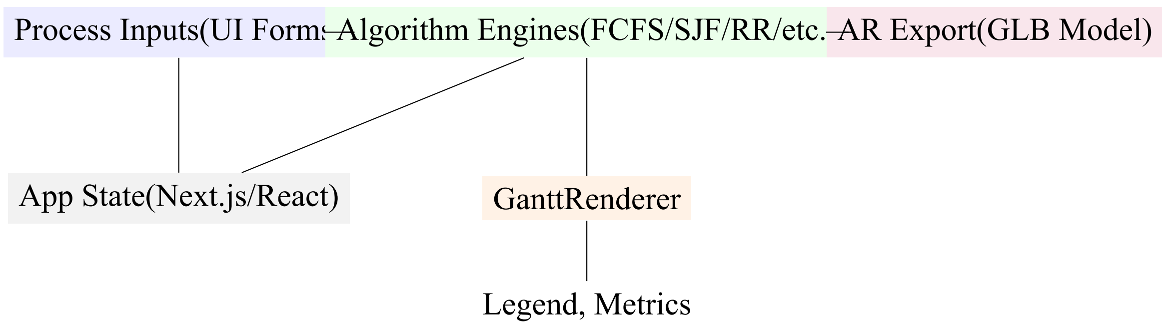

Figure 2.1: High-level block diagram of OSVerse.

2.4 SYSTEM REQUIREMENTS

2.4.1 User Characteristics

Primary users are undergraduate CS students, educators, and enthusiasts. Users are expected to be comfortable with web UIs and possess basic OS concepts.

2.4.2 Software and Hardware Requirements

Table 2.2: Software Requirements

oprule Component	Version/Notes
Node.js	LTS (e.g., 18+), for Next.js build/dev
Package Manager	npm or pnpm supported
Browsers	Chrome/Edge/Brave, Firefox, Safari (latest)
Frameworks	Next.js, React, Tailwind CSS
Libraries	Recharts, Framer Motion, glTF-Transform, Lucide icons

Table 2.3: Hardware Requirements

oprule Device	Recommendation
CPU	Modern x86/ARM laptop/desktop
RAM	8GB+ (dev), 4GB+ (viewing)
GPU	Integrated is sufficient; mobile GPUs for AR
Camera	Required for AR on supported devices

2.4.3 Constraints

- Camera permission (for AR) must be granted by the browser.
- WebXR availability varies by browser/device; AR export via GLB provides a fallback.
- Mobile performance constraints necessitate efficient animations.

2.5 CONCEPTUAL MODELS

2.5.1 Data Flow Diagram

Figure 2.2 shows the Level-0 context diagram. Figure 2.3 expands scheduling computation and visualization.

Figure 2.2: DFD Level-0: OSVerse as a single process.

Figure 2.3: DFD Level-1: Core process breakdown.

2.5.2 ER Diagram

The current system is stateless at runtime, persisting no data. A future enhancement could introduce entities: *Scenario* (metadata), *Process* (arrival, burst, priority), and *Algorithm* (type, parameters). Relationships: Scenario 1..* Process; Scenario 1..1 Algorithm.

3. SYSTEM DESIGN

This chapter details architecture, module and interface design, and data considerations. The system uses a component-driven architecture with clear separation of algorithm computation, visualization, and export.

3.1 SYSTEM ARCHITECTURE

OSVerse uses Next.js 14+ with the App Router. Pages under `src/app/*/page.tsx` provide route-level experiences for each algorithm (e.g., `/fcfs`, `/round-robin`, `/preemptive-sjf`, etc.). Visualization is composed from reusable components (e.g., `GanttChart`, `AnimatedGanttChart`, `SchedulingTemplate`, queue/animation canvases). A serverless API route `/api/gantt-model` builds GLB models using glTF-Transform.

Figure 3.1 illustrates the architecture.

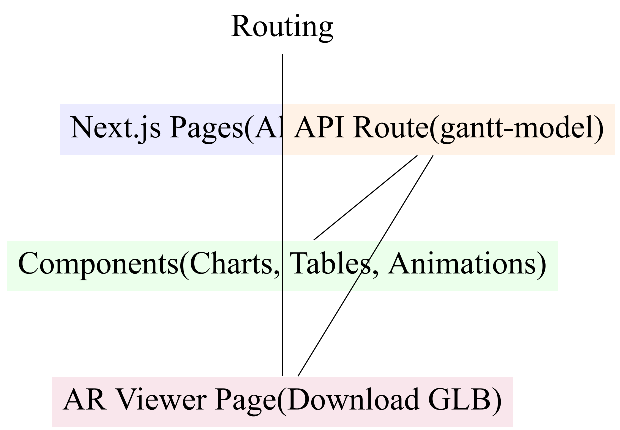

Figure 3.1: System architecture: pages compose components; API exports AR models.

3.2 MODULE DESIGN

Table 3.1: Module Overview

oprule Module	Responsibilities / Key Files
Algorithm Pages	Route-specific UIs, inputs, summaries. E.g., `src/app/fcfs/page.tsx`, `src/app/round-robin/page.tsx`, and others.
Gantt Charts	Static and animated charts. `src/components/GanttChart.tsx`, `AnimatedGanttChart.tsx`.
Process Table	Dynamic process CRUD input. `src/components/ProcessTable.tsx`.
Canvas Animations	Live timeline canvases. `SchedulingAnimation.tsx`, `EnhancedSchedulingAnimation.tsx`.
AR Viewer	Downloads AR GLB for viewing. `src/app/ar-viewer/page.tsx`.
AR Model API	Converts Gantt to GLB. `src/app/api/gantt-model/route.ts`.
UI Kit	Buttons, inputs, forms, cards, tabs under `src/components/ui/*.tsx`.

3.3 DATABASE DESIGN

The current application is client-centric and stateless, persisting no data. A future extension could add persistence for saved scenarios, user profiles, and shared links.

3.3.1 Tables and Relationships

Potential tables: `Users`, `Scenarios`, `Processes`, `Algorithms`. `Scenarios` 1..* `Processes`; `Scenarios` 1..1 `Algorithms`.

3.3.2 Data Integrity and Constraints

Foreign key constraints between `Scenarios` and `Processes`; validations for non-negative arrival/burst; algorithm parameter ranges.

3.4 SYSTEM CONFIGURATION (OPTIONAL)

Environment variables are minimal (none required for basic usage). The project builds with Next.js defaults and can be deployed to Vercel. The API route performs in-process GLB generation using glTF-Transform.

3.5 INTERFACE DESIGN AND PROCEDURAL DESIGN

3.5.1 User Interface Design

The UI emphasizes clarity and progressive disclosure. Each algorithm page uses `SchedulingTempla` to compose a heading, description, input table, controls, result metrics, and the Gantt chart. Components are responsive and adapt to mobile/desktop layouts. Keyboard focus management and contrast-aware colors are prioritized.

3.5.2 Application Flow/Class Diagram

At a high level: (1) user edits processes/parameters; (2) presses Calculate; (3) algorithm engine returns schedule; (4) results and charts update; (5) optional AR export encodes GLB.

3.6 REPORTS DESIGN

The primary "report" is an on-screen summary including per-process finish time, TAT, WT, and their averages. Additionally, an AR model `.glb` can be downloaded for 3D/AR review.

4. IMPLEMENTATION

This chapter summarizes implementation approaches, coding standards, and key code segments. The codebase is TypeScript-first with modular React components and small, testable functions for algorithm logic.

4.1 IMPLEMENTATION APPROACHES

An iterative, component-first approach was adopted:

1. Establish core UI primitives (cards, buttons, inputs) and layout.
2. Implement process input table and a reusable scheduling template.
3. Build algorithm engines incrementally (FCFS, RR, then others).
4. Add static and animated Gantt charts; refine legends and tooltips.
5. Implement AR export API and viewer workflow.
6. Optimize performance and refine accessibility.

4.2 CODING STANDARD

Coding conventions include:

- TypeScript types and interfaces for props and data (e.g., `GanttEntry`).
- React functional components, hooks, and composable props.
- File naming aligned with component names; colocated logic.
- Linting/formatting via ESLint/Prettier defaults for Next.js projects.
- Granular commits with clear messages.

4.3 CODING DETAILS

FCFS Scheduling Calculation

```
function calculateFCFS(processes: Process[]): SchedulingResult
   {
  const sorted = [...processes].sort((a, b) => a.arrival - b.
     arrival);
  let time = 0, totalTAT = 0, totalWT = 0; const gantt:
     GanttEntry[] = [];
  const results = sorted.map((p) => {
    if (time < p.arrival) { time = p.arrival; }
    const start = time; const finish = start + p.burst; time =
       finish;
    const tat = finish - p.arrival; const wt = tat - p.burst;
    totalTAT += tat; totalWT += wt; gantt.push({ name: p.name,
       start, end: finish });
    return { ...p, finish, tat, wt };
  });
  return { results, avgTAT: (totalTAT / processes.length).
     toFixed(2),
           avgWT: (totalWT / processes.length).toFixed(2),
              gantt };
}
```

Listing 4.1: FCFS schedule calculation (excerpt from `src/app/fcfs/page.tsx`).

Round Robin Scheduling

```
function calculateRR(processes: Process[], quantum: number):
   SchedulingResult {
  let time = 0, queue: number[] = [], gantt: GanttEntry[] = [];
  const n = processes.length, rem = processes.map(p => p.burst)
     ;
  const finish = Array(n).fill(0), tat = Array(n).fill(0), wt =
      Array(n).fill(0);
  const arrived = Array(n).fill(false); let completed = 0;
  while (completed < n) {
    for (let i = 0; i < n; i++) if (!arrived[i] && processes[i
       ].arrival <= time) { queue.push(i); arrived[i] = true; }
    if (queue.length === 0) { time++; continue; }
    const idx = queue.shift()!; const exec = Math.min(quantum,
       rem[idx]);
```

```
    gantt.push({ name: processes[idx].name, start: time, end:
       time + exec });
    time += exec; rem[idx] -= exec;
    for (let i = 0; i < n; i++) if (!arrived[i] && processes[i
       ].arrival <= time) { queue.push(i); arrived[i] = true; }
    if (rem[idx] > 0) queue.push(idx); else { finish[idx] =
       time; tat[idx] = finish[idx] - processes[idx].arrival;
       wt[idx] = tat[idx] - processes[idx].burst; completed++;
       }
  }
  const totalTAT = tat.reduce((a, b) => a + b, 0), totalWT = wt
     .reduce((a, b) => a + b, 0);
  return { results: processes.map((p, i) => ({ ...p, finish:
     finish[i], tat: tat[i], wt: wt[i] })),
           avgTAT: (totalTAT / n).toFixed(2), avgWT: (totalWT /
               n).toFixed(2), gantt };
}
```

Listing 4.2: Round Robin calculation (excerpt from `src/app/round-robin/page.tsx`).

Animated Gantt Rendering

```
const AnimatedGanttChart: React.FC<AnimatedGanttChartProps> =
    ({ gantt, colorScheme, algorithm = "Scheduling" }) => {
  const [currentTime, setCurrentTime] = useState(0);
  const [isPlaying, setIsPlaying] = useState(false);
  const maxTime = gantt.length > 0 ? Math.max(...gantt.map((g)
     => g.end)) : 10;
  const uniqueProcesses = Array.from(new Set(gantt.filter(g =>
     g.name !== "Idle").map(g => g.name)));
  // ... playback controls and colored cells per time unit ...
};
```

Listing 4.3: Animated Gantt chart (excerpt from `src/components/AnimatedGanttChart.tsx`).

AR Export API

```
export async function GET(req: NextRequest) {
  const gantt = JSON.parse(req.nextUrl.searchParams.get('data')
     !);
  const doc = new Document(); doc.createBuffer(); const scene =
      doc.createScene('Gantt');
```

```
  const maxTime = Math.max(...gantt.map((e: any) => e.end));
  const textMaterial = doc.createMaterial('Text').
     setBaseColorFactor([0.05,0.05,0.05,1]);
  // Create bars per gantt entry as rectangular meshes; add
     labels and markers
  // Serialize to GLB
  const io = new NodeIO(); const glb = await io.writeBinary(doc
     );
  return new NextResponse(new Uint8Array(glb), { status: 200,
     headers: { 'Content-Type': 'model/gltf-binary' } });
}
```

Listing 4.4: API route generating GLB from Gantt data (excerpt from `src/app/api/gantt-model/route.ts`).

4.4 SCREEN SHOTS

Figures for representative algorithm pages, animated Gantt charts, and AR download UI should be inserted close to the relevant text when available. Captions should follow the specified 10pt style.

5. TESTING

This chapter presents test cases, approaches, and reports. Testing emphasizes algorithm correctness, UI behavior, and AR export robustness.

5.1 TEST CASES

Table 5.1: Representative Test Scenarios and Expected Outcomes

oprule Scenario	Expected Outcome
FCFS with staggered arrivals	Processes execute in arrival order; no preemption; correct WT/TAT.
Round Robin with quantum=2	Time-sliced execution; fairness across processes; correct totals.
Priority (preemptive)	Higher priority interrupts lower; prevent starvation using aging (future).
SJF (non-preemptive)	Shortest job first without preemption; correct total waiting time reduction.
AR Export	GLB downloads successfully; can be viewed in device AR viewer.

5.2 TESTING APPROACHES

- **Unit tests (future addition):** Algorithm engines validated against known examples.
- **Component tests:** ProcessTable input validation; Gantt render with empty/edge cases.
- **Integration tests:** Pages render and compute results; AR route returns GLB for valid input.

5.3 TEST REPORTS

Sample FCFS input: P1(0,4), P2(2,3), P3(6,2). Observed output: Avg TAT and WT match computed values; Gantt blocks appear in sequence with idle gaps if any. Round

Robin sample: four processes with quantum=2 yield interleaved bars and expected averages.

6. CONCLUSION

This chapter revisits objectives, summarizes features and results, notes limitations, and outlines future work.

6.1 DESIGN AND IMPLEMENTATION ISSUES

Key challenges included:

- **Animation performance:** Balancing DOM-based grids vs. Canvas renderers and optimizing rerenders.
- **Color/contrast:** Ensuring readability and accessibility across dark/light themes.
- **AR export fidelity:** Creating legible 3D text and bars with the glTF pipeline without heavy dependencies.
- **Cross-browser quirks:** Minor differences in event timing and font rendering.

6.2 ADVANTAGES AND LIMITATIONS

extbfAdvantages: Web-native, no install; broad algorithm coverage; interactive animations; AR augmentation; modular architecture.

extbfLimitations: No backend persistence yet; limited automated tests; AR support varies by device; advanced algorithms (e.g., EDF/MLFQ) require rigorous validation for edge cases.

6.3 FUTURE SCOPE OF THE PROJECT

Potential enhancements include saved scenarios and sharing, collaborative editing, printable PDF reports, accessibility audits, a11y annotations in charts, additional OS modules (deadlocks, memory management), and expanded AR interactions.

A. PROJECT STRUCTURE

The following structure summarizes the repository at the time of writing:

```
ANIMATION_README.md
AR_README.md
components.json
ENHANCED_ANIMATION_README.md
next-env.d.ts
next.config.ts
package.json
postcss.config.mjs
README.md
tailwind.config.js
tsconfig.json
public/
  file.svg
  globe.svg
  next.svg
  scheduling-preview.svg
  vercel.svg
  window.svg
src/
  app/
    favicon.ico
    globals.css
    layout.tsx
    page.tsx
    algorithms/
      page.tsx
    api/
      gantt-model/
        route.ts
    ar-demo/
```

```
      page.tsx
    ar-viewer/
      page-new.tsx
      page-old.tsx
      page.tsx
    edf/
      page.tsx
    fair-share/
      page.tsx
    fcfs/
      page.tsx
    hrrn/
      page.tsx
    ljf/
      page.tsx
    lottery/
      page.tsx
    multilevel-feedback-queue/
      page.tsx
    multilevel-queue/
      page.tsx
    preemptive-priority/
      page.tsx
    preemptive-sjf/
      page.tsx
    priority/
      page.tsx
    round-robin/
      page.tsx
    sjf/
      page.tsx
    view-model/
      page.tsx
  components/
    AlgorithmCard.tsx
    AnimatedGanttChart.tsx
    BackgroundAnimation.tsx
    ClassicGanttChart.tsx
    EnhancedARModal.tsx
```

```
    EnhancedQueueAnimation.tsx
    EnhancedSchedulingAnimation.tsx
    EnhancedVideoAnimation.tsx
    GanttChart.tsx
    LazyAnimation.tsx
    ProcessTable.tsx
    QueueAnimation.tsx
    SchedulingAnimation.tsx
    SchedulingTemplate.tsx
    SimpleARModal.tsx
    StreamlinedARModal.tsx
    VideoAnimation.tsx
    WebXRManager.tsx
    ui/
      button.tsx
      card.tsx
      form.tsx
      input.tsx
      label.tsx
      tabs.tsx
  lib/
    ar-service.ts
    utils.ts
  types/
    model-viewer.d.ts
    webxr.d.ts
```

B. USER MANUAL

ACCESSING THE APPLICATION

Open https://osverse.shreyjain.me. Navigate via the Home page or the Algorithms index.

RUNNING AN ALGORITHM

1. Select an algorithm route (e.g., FCFS, Round Robin).
2. Use the process table to add rows and set arrival/burst/priority.
3. Click *Calculate Results*. Review per-process results and averages.
4. Use the animated Gantt to play/pause and adjust speed.

EXPORTING AR MODEL

On supported pages, choose AR export or navigate to the AR Viewer. Provide or carry forward the Gantt data and download the `.glb`. Use your device's AR viewer to place and inspect the 3D timeline.

TROUBLESHOOTING

If AR export fails, ensure you have stable network and a supported browser. For performance issues, reduce the number of processes or animation speed.

C. PLANNING ARTIFACTS

Development proceeded in iterative milestones: core UI, FCFS baseline, RR with animation, expanded algorithm coverage, AR export, and UX polish. Future sprints target persistence, collaboration, and test automation.

BIBLIOGRAPHY

[1] Roger S. Pressman. Software Engineering: A Practitioner's Approach. 7th ed. McGraw-Hill Education, 2010.

[2] W3C. "WebXR Device API." Latest Working Draft. Accessed: August 20, 2025. https://www.w3.org/TR/webxr/.

[3] Vercel. "Next.js Documentation." Accessed: August 20, 2025. https://nextjs.org/docs.

[4] Meta. "React Documentation." Accessed: August 20, 2025. https://react.dev.

[5] Microsoft. "TypeScript Documentation." Accessed: August 20, 2025. https://www.typescriptlang.org/docs/.

[6] Tailwind Labs. "Tailwind CSS Documentation." Accessed: August 20, 2025. https://tailwindcss.com/docs.

[7] Recharts. "Recharts Documentation." Accessed: August 20, 2025. https://recharts.org/en-US/.

[8] Framer. "Framer Motion." Accessed: August 20, 2025. https://www.framer.com/motion/.

[9] Khronos Group. "glTF 2.0 Specification." Accessed: August 20, 2025. https://www.khronos.org/gltf/.

[10] Don McCurdy. "glTF-Transform." Accessed: August 20, 2025. https://gltf-transform.donmccurdy.com/.

[11] Vercel. "Vercel Platform." Accessed: August 20, 2025. https://vercel.com/.