499 Project Requirements

**Target User Group:**

* Graduating students or early professionals who rely on their computer for academic, professional, or creative work
* Users who want to consolidate their projects and are preparing to showcase their skills to potential employers/clients/graduate schools.

**Usage Scenario:**

* Generate/Update resume or portfolio to share work with public

They hope to use this information to showcase as part of a web portfolio, or a dashboard of some kind that lets them easily see the metrics and highlights of their hard work, or descriptions and metrics that could be useful for improving their résumeé.

* User may be applying for positions and needs a quick review of their contributions and work
* User may be interested in reviewing the artifacts that exist on their device without sharing any data/information externally
* Generate overall score for user query, for example, if user asks to judge their eloquence in their writing of essays, we can utilize the parser to give them advice and how to improve their speech

**Functional:**

1. **Artifact Discovery/Indexing:** Scans across all important artifact locations that the user has selected (Github, Word, etc…)
   1. Once user gives permission the tool will scan the directory and segregate files based on file types
   2. Stores the metadata for each file after scanning the directory - filename, type, creation date, last modified and file size

*Test: User selects a directory and the software successfully lists all files with the correct file types and metadata*

1. **Artifact Analysis:** Software detects information from different file types and processes them accordingly
   1. Detects programming languages based on file extension, if Git Repo, count commits and extract first and last commit dates
   2. Extract metadata from document files - for example word count from a word document and last edited dates from similar information files (PDF, word, excel etc)
   3. Extract resolution (for images), duration of file (for audio/video)

*Test: User scans a folder containing such different files and software displays above information - commits, resolution, word count etc*

1. **Insights and Summaries**: (Maybe) Shows the overall count for different artifacts based on types and a graph of timeline based on creation or last edited dates

*Test: If implemented - it would be through manual testing of the frontend/UI to check if dashboard gets updated when new artifacts are added or existing ones are updated*

1. **Data Privacy and Control:** Allows users to take control of what files and data is visible to the software and is being used
   1. User can exclude artifacts to control data visibility
   2. User is able to delete previously indexed data from memory

*Test: User excludes a particular folder and the insight and summary information is removed; user deletes previous data and it disappears from frontend dashboard*

1. ***Configuration Management:***Allows users to create and manage custom scanning settings and preferences
2. User can create and save multiple scanning profiles with different file type priorities and exclusion rules
3. System remembers user preferences and settings between application sessions
4. Users can set default directories and scanning parameters for quick access

*Test: User creates a "Work Projects" profile, saves it, restarts application, and profile loads with correct settings*

***6. Duplicate Detection & Management:***System identifies and manages duplicate files to provide accurate metrics

1. System detects duplicate files across different directories using content hash comparison
2. Users can view duplicate file lists and choose which instances to include in analysis
3. Syste*m* flags potential duplicate projects (same Git repository in multiple locations)

*Test: User has same document in two folders; system identifies duplicates and allows user to exclude one from metrics*

***7. Project Grouping & Organization:*** System automatically organizes files into logical project groups

1. System groups related files based on directory structure and Git repository boundaries
2. Users can manually create, edit, and merge project groups
3. System suggests project names based on folder names and repository information

*Test: Files within same Git repository are automatically grouped, user can split group or rename project*

***8. Export & Reporting:***System generates formatted reports and data exports for portfolio use

1. System creates PDF and HTML reports with artifact summaries and key metrics
2. Users can export filtered data for specific time periods, file types, or projects
3. System generates JSON exports for integration with external portfolio tools

*Test: User exports 6-month coding activity report and receives properly formatted PDF with charts and statistics*

***9. Search & Filtering:*** System provides advanced search and filtering capabilities for artifact discovery

1. Users can search artifacts by filename, project name, programming language, or date range
2. System offers filtering by file size, modification date, and custom metadata tags
3. Users can save frequently used search queries for quick access

*Test: User searches for "Python files modified in last 30 days larger than 1KB" and receives accurate filtered results*

**Non-Functional: (all points can be modified/removed if we don't agree on something)**

1. **Performance:**
   1. The system shall scan and index 1 GB of data within 5 seconds on a standard machine (8 G RAM, SSD)
   2. The system shall scan and index up to 1000 files within 2 minutes on a standard machine (8GB RAM, SSD)
   3. The system shall support incremental scanning (only process new/modified files) to reduce subsequent scan times by 50%
   4. The dashboard shall load and display artifact lists within 3 seconds
   5. All user interactions (sorting, viewing, filtering details) shall respond within 2 seconds
2. **Scalability & Capacity:**
3. The system shall handle directories containing up to 10,000 files without performance degradation
4. The system shall support analyzing Git repositories with up to 5,000 commits within 30 seconds
5. The system shall maintain responsive performance when analyzing individual files up to 100 MB in size
6. **Reliability & Error Handling:**
7. The system shall gracefully handle file access permission errors and continue scanning remaining files
8. The system shall recover from interrupted scans and resume from the last processed file
9. The system shall maintain 99% uptime during normal operation without crashes or data corruption
10. **Compatibility & Portability:**
11. The system shall run on Windows 10+, macOS 10.15+, and Ubuntu 18.04+ operating systems
12. The system shall work with file systems using UTF-8, UTF-16, and ASCII character encodings
13. The system shall support Git repositories created with Git versions 2.0 and above
14. **Security & Data Integrity:**
15. The system shall not transmit any user data over network connections
16. The system shall store all indexed data using local file encryption when at rest
17. The system shall validate file integrity using checksums before processing
18. **Usability & Accessibility:**
19. The system shall provide clear error messages with suggested solutions for common issues
20. The command-line interface shall support standard keyboard shortcuts and tab completion
21. The system shall complete initial setup and first scan within 5 minutes for a typical user

[Condensed Work Breakdown Structure (WBS)](#_vl76efkkigol)

[Phase 1: Foundation (Weeks 1-4)](#_r2txyjx6le5k)

[Phase 2: Analysis Engine (Weeks 5-8)](#_cpe29jws1r8q)

[Phase 3: User Interface & Privacy (Weeks 9-12)](#_f247i0xs5gw2)

[Phase 4: Testing & Documentation (Weeks 13-16)](#_nek1lxogw22)

[Key Dependencies:](#_vfyoq3huj1ox)

## **Condensed Work Breakdown Structure (WBS)**

### 

### **Phase 1: Foundation (Weeks 1-4)**

**1.1 Problem Domain Research**

* Literature review on digital artifact mining and personal productivity analytics
* Competitive analysis of existing portfolio builders and productivity tools
* Research on privacy-preserving personal data analysis techniques
* Study of artifact metadata extraction methodologies

**1.2 User Research & Requirements Gathering**

* Conduct user interviews with graduating students and early professionals (6-8 participants)
* Develop detailed user personas and journey maps
* Define functional requirements through user story mapping
* Establish non-functional requirements (performance, security, scalability)

**1.3 User Research & Requirements Gathering**

* Define system boundaries and integration points
* Identify technical constraints and limitations
* Establish privacy and ethical guidelines for personal data handling

**1.4 Use Case Development & Validation**

* Develop comprehensive use case diagrams and scenarios
* Create detailed user acceptance criteria
* Design system context diagram and stakeholder analysis
* Validate requirements through stakeholder review sessions

**Deliverable**: Requirements Specification Document with user research findings and system scope

## **Phase 2: System Architecture Design**

### **2.1 Technology Stack Architecture**

* Next.js application architecture design (App Router vs Pages Router)
* NeonDB database architecture and connection strategy
* Prisma ORM integration and schema design approach
* Cloud deployment architecture (Vercel platform considerations)

### **2.2 System Component Architecture**

* High-level system component diagram and interaction flows
* Microservice vs monolithic architecture decision analysis
* API architecture design with RESTful endpoint planning
* Frontend component hierarchy and state management architecture

### **2.3 Data Flow & Processing Architecture**

* File upload and processing pipeline architecture
* Background job processing design (queues, workers, scheduling)
* Real-time data synchronization strategies
* Caching architecture (client-side, server-side, database)

### 

### **2.4 Security & Privacy Architecture**

* Authentication and authorization framework design
* Data encryption strategies (at rest and in transit)
* Privacy-by-design implementation architecture
* API security and rate limiting mechanisms

**Deliverable**: System Architecture Document with component diagrams and technology decisions

## **Phase 3: Database & API Design**

### **3.1 Database Schema Design**

* Entity-relationship modeling for artifacts, projects, users, and metadata
* NeonDB-specific schema optimization and indexing strategies
* Data normalization analysis and relationship mapping
* Database migration and versioning strategy

### **3.2 API Specification Design**

* RESTful API endpoint design with resource modeling
* OpenAPI/Swagger specification development
* API versioning and backward compatibility strategy
* Request/response data models and validation schemas

### **3.3 Data Processing Algorithm Design**

* File analysis algorithms for different artifact types:
  + - Git repository metadata extraction and commit analysis
    - Document parsing (PDF, DOCX, TXT) and word count algorithms
    - Media file processing (image resolution, video duration)
    - Programming language detection and code analysis
* Background processing workflow design

### **3.4 Analytics & Reporting Architecture**

* Metrics calculation and aggregation algorithms
* Time-series data handling for productivity analytics
* Dashboard data preparation and caching strategies
* Export functionality design (JSON, PDF, CSV formats)

**Deliverable**: Database Design Specification and API Documentation with processing algorithms

## **Phase 4: User Interface & Integration Architecture**

### **4.1 Frontend Architecture Design**

* Next.js component architecture and page structure
* State management strategy (React Context, Zustand, or Redux Toolkit)
* Client-side routing and navigation architecture
* Progressive Web App (PWA) capabilities and offline functionality

### **4.2 User Experience Architecture**

* Information architecture and site map design
* User interface wireframes and interaction flows
* Responsive design system architecture
* Accessibility compliance framework (WCAG 2.1)

### **4.3 Data Visualization & Dashboard Design**

* Chart and visualization component architecture
* Real-time data update mechanisms
* Interactive dashboard layout and widget system
* Mobile-responsive dashboard design patterns

### **4.4 Integration & Deployment Architecture**

* External service integration design (GitHub API, file system access)
* CI/CD pipeline architecture for Next.js and NeonDB
* Environment management strategy (development, staging, production)
* Monitoring, logging, and error tracking architecture

**Deliverable**: Frontend Architecture Specification and Integration Design Document

### **Key Dependencies:**

* File system access across different operating systems
* Git repository parsing reliability
* Performance optimization for large datasets