**Backend**python –m backend.api.run  
  
**Frontend**npm start  
  
**QUAST**root@Anarchy:/mnt/c/Users/Anarchy/Documents/Data\_Science/genome\_work/genome\_filtering\_tool# python3 tools/quast/quast.py\ cholera/cholera/18\_23\_scaffolds.fasta \ data/results/d47af17a\_18\_23\_scaffolds\_filtered\_6f206f29.fasta \ -r cholera/reference.fna \ -o quast\_results\_with\_ref \ --gene-finding \ --threads 4

**QUAST Integration Plan for Genome Filtering Tool (Windows/WSL)**

**Phase 1: WSL Communication Layer**

**Step 1: Create Path Conversion Utilities**

1. Create a new file backend/utils/wsl\_path\_converter.py:
   * Implement Windows-to-WSL path conversion
   * Implement WSL-to-Windows path conversion
   * Add validation to ensure paths exist

**Step 2: Create WSL Command Execution Utilities**

1. Create a new file backend/utils/wsl\_executor.py:
   * Implement a function to execute commands in WSL
   * Add timeout handling for large genomes
   * Add proper error handling
   * Implement output capture and parsing

**Phase 2: QUAST Analysis Core**

**Step 3: Create QUAST Analysis Module**

1. Create a new file backend/core/quast\_analysis.py:
   * Implement function to run QUAST on one or more assemblies
   * Create functions to parse QUAST's TSV/JSON outputs
   * Design a schema for storing QUAST results
   * Add utility to extract key metrics from QUAST reports

**Step 4: Create QUAST Result Parser**

1. Create a new file backend/core/quast\_parser.py:
   * Implement parsers for report.tsv, transposed\_report.tsv
   * Extract metrics like N50, L50, total length, # contigs, etc.
   * Parse alignment information when reference genomes are used
   * Generate simplified metric summaries for display

**Phase 3: Integration with Workflow**

**Step 5: Update Filtering Workflow**

1. Modify [workflow.py](vscode-file://vscode-app/c:/Users/Anarchy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html):
   * Add QUAST analysis step after filtering
   * Pass both original and filtered assemblies to QUAST
   * Add configuration options for QUAST run
   * Ensure QUAST results are saved with filtering results

**Step 6: Update Job Storage Structure**

1. Modify the job directory structure:
   * Create quast/ subdirectory in results folder
   * Store QUAST reports, generated plots and HTML
   * Create metadata file linking QUAST results with filtering job

**Phase 4: API Updates**

**Step 7: Create QUAST API Models**

1. Update [models.py](vscode-file://vscode-app/c:/Users/Anarchy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html):
   * Create QuastOptions, QuastResults models
   * Add QUAST parameters (reference genome, gene finding, etc.)
   * Update FilterResultResponse to include QUAST metrics

**Step 8: Add QUAST API Endpoints**

1. Update [main.py](vscode-file://vscode-app/c:/Users/Anarchy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html):
   * Add GET endpoint to retrieve QUAST results
   * Add endpoint to download QUAST HTML report
   * Update existing results endpoint to include QUAST metrics
   * Add endpoint to upload reference genomes (optional)

**Phase 5: Frontend Updates**

**Step 9: Add QUAST Types**

1. Update [api.ts](vscode-file://vscode-app/c:/Users/Anarchy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "):
   * Add QuastMetric interface
   * Add QuastResults interface
   * Update FilterResults interface to include QUAST

**Step 10: Create QUAST API Service**

1. Update [api.ts](vscode-file://vscode-app/c:/Users/Anarchy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "):
   * Add function to fetch QUAST results
   * Add function to get QUAST report URL
   * Add function to upload reference genomes (optional)

**Step 11: Create QUAST Results Components**

1. Create new frontend components:
   * frontend/src/components/QuastResultsTab.tsx
   * frontend/src/components/QuastMetricsCard.tsx
   * frontend/src/components/QuastComparisonChart.tsx

**Step 12: Update Results Page**

1. Modify [ResultsPage.tsx](vscode-file://vscode-app/c:/Users/Anarchy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "):
   * Add "QUAST Analysis" tab
   * Integrate QUAST result components
   * Add links to full HTML reports
   * Show visual comparison of key metrics

**Phase 6: Configuration Options**

**Step 13: Add QUAST Configuration Options**

1. Update [FilterMethodSelector.tsx](vscode-file://vscode-app/c:/Users/Anarchy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "):
   * Add section for QUAST options
   * Add reference genome upload option
   * Add gene finding option
   * Add other QUAST parameters

**Step 14: Update Configuration Validator**

1. Update [config\_validator.py](vscode-file://vscode-app/c:/Users/Anarchy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html):
   * Add validation for QUAST parameters
   * Handle reference genome paths
   * Verify QUAST options are valid

**Phase 7: Testing and Refinement**

**Step 15: Create Test Cases**

1. Create tests in backend/tests/test\_quast.py:
   * Test WSL command execution
   * Test path conversion
   * Test QUAST result parsing
   * Test integration with workflow

**Step 16: Add Progress Reporting**

1. Update both backend and frontend:
   * Add progress reporting for QUAST analysis
   * Show progress in frontend
   * Handle timeout/errors gracefully

**Step 17: Optimize for Performance**

1. Implement performance improvements:
   * Add caching for QUAST results
   * Add option to skip QUAST for large assemblies
   * Optimize path conversions for frequent operations

**Phase 8: Documentation and Polish**

**Step 18: Update Documentation**

1. Add documentation for QUAST integration:
   * Document how to configure QUAST options
   * Explain metrics shown in results
   * Add tutorial for using reference genomes

**Step 19: Add Error Handling**

1. Improve error handling:
   * Handle missing QUAST installation
   * Provide clear error messages for WSL issues
   * Handle timeout errors for large assemblies

**Step 20: Add Final Polish**

1. Final integration improvements:
   * Add QUAST version checking
   * Add option to save/export QUAST results
   * Add bookmarking/sharing of QUAST results

By following this phased approach, you'll be able to seamlessly integrate QUAST with your genome filtering tool, allowing users to immediately see quality assessment metrics for their filtered assemblies without leaving your application.