# ID608001: Intermediate Application Development Concepts

# Project Marking Rubric

Functionality - Learning Outcomes 1 and 2 (55%)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A – 10-8 | B – 7.5-6.5 | C – 6-5 | D/E – 4.5-0 |
| Milestone One - Movie Listings Application (10%) | Perfectly structured Vite React project in the specified directory. Successfully deployed to Vercel with proper configuration.  Excellent implementation of React Query/TanStack Query and Fetch API for data retrieval from The Movie DB API.  Perfect implementation of navigation bar using Tailwind CSS, Shadcn UI, and React Router with all specified movie type options (Trending, Top Rated, Action, Animation, Comedy). Default selection is properly set to Trending.  Flawless display of title, overview, poster path, and release date for the first 10 movies in each category. Perfect card styling using Tailwind CSS and Shadcn UI. Exactly five movies per row in a responsive layout.  Comprehensive error handling for all potential error states and missing data scenarios. Intuitive error messages and graceful fallbacks. | Well-structured Vite React project in the specified directory. Successfully deployed to Vercel with minor configuration issues  Good implementation of React Query/TanStack Query and Fetch API with proper data retrieval.  Good implementation of navigation bar with all specified movie type options. Default selection works correctly.  Good display of movie information with appropriate styling. Five movies per row with minor layout issues.  Good error handling for most error states and missing data scenarios. | Basic Vite React project in the specified directory. Deployed to Vercel with some significant issues.  Basic implementation of React Query/TanStack Query and Fetch API with functional data retrieval.  Basic navigation bar that includes all options but with styling or functional issues.  Basic display of movie information with minimal styling. Approximately five movies per row with noticeable layout issues.  Basic error handling that catches major errors but misses edge cases or lacks graceful handling. | Poorly structured or incomplete Vite React project. Deployment to Vercel unsuccessful or with major issues.  Poor or incomplete implementation of React Query/TanStack Query and Fetch API. Data retrieval is unreliable or broken.  Incomplete navigation bar missing options or with significant functional issues.  Incomplete or poorly styled movie display. Row layout incorrect or inconsistent.  Minimal or no error handling for missing data or API failures. |
|  | A – 15-12 | B – 11.5-10 | C – 9.5-7.5 | D/E – 7-0 |
| Milestone Two - Hacker News Application (15%) | Perfectly structured Vite React project in the specified directory. Successfully deployed to Vercel with proper configuration.  Excellent implementation of React Query/TanStack Query and Fetch API for data retrieval from the Hacker News API.  Perfect implementation of navigation bar using Tailwind CSS, Shadcn UI, and React Router with all specified story options (Ask Stories, Best Stories, Job Stories, New Stories, Show Stories, Top Stories, Leaders). Default selection properly set to Ask Stories.  Flawless display of titles for the first 25 stories in each category. Perfect card styling using Tailwind CSS and Shadcn UI. Exactly five stories per row in a responsive layout.  Perfect implementation of story detail page with all required information (By, Kids, Score, Time, Title, Type, URL) properly displayed and formatted. All links work correctly and open in new tabs. Time correctly converted to readable format.  Excellent implementation of leader search functionality with proper form validation and hardcoded leader data. Complete leader information display (About, Created, Id, Karma, Submitted) with perfect styling and formatting. Time correctly converted to readable format.  Comprehensive error handling for all potential error states and missing data scenarios. Intuitive error messages and graceful fallbacks. | Well-structured Vite React project in the specified directory. Successfully deployed to Vercel with minor configuration issues.  Good implementation of React Query/TanStack Query and Fetch API with proper data retrieval.  Good implementation of navigation bar with all specified story options. Default selection works correctly.  Good display of story titles with appropriate styling. Five stories per row with minor layout issues.  Good implementation of story detail page with all required information displayed. Links work correctly. Time conversion is functional with minor formatting issues.  Good implementation of leader search functionality with basic validation. Leader information display includes all required fields with appropriate styling.  Good error handling for most error states and missing data scenarios. | Basic Vite React project in the specified directory. Deployed to Vercel with some significant issues.    Basic implementation of React Query/TanStack Query and Fetch API with functional data retrieval.  Basic navigation bar that includes all options but with styling or functional issues.  Basic display of story titles with minimal styling. Approximately five stories per row with noticeable layout issues.  Basic implementation of story detail page with most required information. Some links may not work correctly or time conversion has significant formatting issues.  Basic implementation of leader search with minimal validation. Leader information display missing some fields or with significant styling issues.  Basic error handling that catches major errors but misses edge cases or lacks graceful handling. | Poorly structured or incomplete Vite React project. Deployment to Vercel unsuccessful or with major issues.  Poor or incomplete implementation of React Query/TanStack Query and Fetch API. Data retrieval is unreliable or broken.  Incomplete navigation bar missing options or with significant functional issues.  Incomplete or poorly styled story display. Row layout incorrect or inconsistent.  Missing or severely incomplete story detail page. Multiple required fields missing or non-functional links.  Missing or non-functional leader search. Incomplete leader information display.  Minimal or no error handling for missing data or API failures. |
|  | A – 20-16 | B – 15.5-13 | C – 12.5-10 | D/E – 9.5-0 |
| Milestone Three - Trivia Application (20%) | Perfectly structured Vite React project in the specified directory.  Excellent implementation of React Query/TanStack Query and Fetch API for data retrieval from the OpenTDB API.  Perfect implementation of all required form inputs (Name, Amount, Category, Difficulty, Type) with correct default values. Category options correctly fetched from API endpoint. All form elements properly styled with Tailwind CSS and Shadcn UI.  Flawless implementation of quiz questions and answer options based on API response. Perfectly styled quiz interface with excellent user experience.  Excellent implementation of answer submission form with proper validation. Perfect display of user's name, score, and correct answers after submission.  Perfect implementation of localStorage to store user data (name, score, category) in an array of objects. Data persists correctly between sessions.  Outstanding implementation of leaderboard displaying top 5 scores for each category stored in localStorage. Perfectly styled with Tailwind CSS and Shadcn UI.  Comprehensive error handling for all potential error states and missing data scenarios. Intuitive error messages and graceful fallbacks. | Well-structured Vite React project in the specified directory.  Good implementation of React Query/TanStack Query and Fetch API with proper data retrieval.  Good implementation of all required form inputs with correct default values. Category options correctly fetched. Good styling with minor issues.  Good implementation of quiz questions and answer options. Well-styled interface with good user experience.  Good implementation of answer submission with basic validation. Clear display of results after submission.  Good implementation of localStorage for user data. Data generally persists correctly between sessions with minor issues.  Good implementation of leaderboard showing top scores by category. Well-styled with minor issues.  Good error handling for most error states and missing data scenarios. | Basic Vite React project in the specified directory.  Basic implementation of React Query/TanStack Query and Fetch API with functional data retrieval.  Basic implementation of required form inputs with some issues in default values or category fetching. Minimal styling.  Basic implementation of quiz questions and answers. Functional but with minimal styling or user experience considerations.  Basic implementation of answer submission. Results displayed but with limited information or formatting.  Basic implementation of localStorage with some issues in data structure or persistence.  Basic implementation of leaderboard that shows scores but may have issues with display.  Basic error handling that catches major errors but misses edge cases or lacks graceful handling. | Poorly structured or incomplete Vite React project.  Poor or incomplete implementation of React Query/TanStack Query and Fetch API. Data retrieval is unreliable or broken.  Incomplete form missing required inputs or with non-functional elements.  Incomplete or poorly implemented quiz display. Questions or answers may not display correctly.  Missing or severely limited answer submission functionality. Results not displayed properly.  Missing or non-functional localStorage implementation.  Missing or severely limited leaderboard functionality.  Minimal or no error handling for missing data or API failures. |
|  | A – 10-8 | B – 7.5-6.5 | C – 6-5 | D/E – 4.5-0 |
| Milestone Four - Emerging Technologies Application (10%) | Perfectly structured Astro or Solid.js project in the specified directory.  Flawless implementation of all required todo features (add, delete, edit, mark complete/incomplete, view all/completed/incomplete todos). All features work perfectly with excellent user experience.  Exceptional styling using Tailwind CSS. Interface is attractive, intuitive, and fully responsive. Perfect implementation of layout, typography, colours, and spacing.  Comprehensive error handling for all potential error states and missing data scenarios. Intuitive error messages and graceful fallbacks. | Well-structured Astro or Solid.js project in the specified directory.  Good implementation of all required todo features. All features work correctly with minor issues in user experience.  Good styling using Tailwind CSS. Interface is attractive and functional with minor inconsistencies. Good implementation of layout and design elements.  Good error handling for most error states and missing data scenarios. | Basic Astro or Solid.js project in the specified directory.  Basic implementation of required todo features with some functional limitations or usability issues.  Basic styling using Tailwind CSS. Interface is functional but lacks polish or has significant inconsistencies.  Basic error handling that catches major errors but misses edge cases or lacks graceful handling. | Poorly structured or incomplete Astro or Solid.js project.  Incomplete implementation of todo features with multiple missing or non-functional requirements.  Minimal or poor styling with Tailwind CSS. Interface is unattractive or difficult to use.  Minimal or no error handling for errors or edge cases. |

Code Quality and Best Practices - Learning Outcome 1 (40%)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A – 10-8 | B – 7.5-6.5 | C – 6-5 | D/E – 4.5-0 |
| Project Structure (10%) | Exceptional project structure with clear separation of concerns. Files and directories are logically organised with consistent patterns. Components, hooks, utilities, and assets are perfectly separated.  Code is highly modular with excellent reusability.  Dependencies are well-managed with clear import/export patterns.  Structure demonstrates expert understanding of React/frontend architecture principles across all milestone applications. | Good project structure with clear separation of concerns. Logical organisation of files and directories with minor inconsistencies.  Good modularity and reusability.  Well-managed dependencies with occasional minor issues in import/export patterns.  Structure demonstrates good understanding of React/frontend architecture principles in most milestone applications. | Acceptable project structure with basic separation of concerns. Some logical organisation but with notable inconsistencies.  Limited modularity and reusability.  Dependencies managed but with some significant issues.  Structure demonstrates basic understanding of React/frontend architecture principles with inconsistent application across milestone applications. | Poor project structure with minimal separation of concerns. Disorganised files and directories with major inconsistencies.  Little to no modularity or reusability.  Poorly managed dependencies with critical issues.  Structure demonstrates limited understanding of React/frontend architecture principles in most or all milestone applications. |
|  | A – 5-4 | B – 3.5 | C – 3-2.5 | D/E – 2-0 |
| Naming Conventions (5%) | Exceptional naming conventions throughout the codebase.  All files, components, functions, and variables have clear, descriptive, and consistent names that perfectly reflect their purpose.  Naming follows React/JS industry standard conventions (PascalCase for components, camelCase for functions/variables) appropriately throughout all code.  Naming demonstrates exceptional clarity and consistency across all milestone applications. | Good naming conventions with minor inconsistencies.  Most files, components, functions, and variables have clear and descriptive names.  Naming mostly follows appropriate conventions with occasional deviations.  Good overall clarity and consistency in naming across most milestone applications. | Acceptable naming conventions but with notable inconsistencies.  Some files, components, functions, and variables lack clarity or descriptiveness.  Inconsistent application of naming conventions.  Basic level of clarity in naming with significant room for improvement across multiple milestone applications. | Poor naming conventions throughout.  Many files, components, functions, and variables have unclear, non-descriptive, or misleading names.  Little to no consistent application of naming conventions.  Names create confusion and hinder code readability in most milestone applications. |
|  | A – 5-4 | B – 3.5 | C – 3-2.5 | D/E – 2-0 |
| Documentation and Comments (5%) | Documentation using JSDoc consistently maintained across all milestone applications.  All components, functions, hooks, and complex sections have comprehensive, clear comments that perfectly explain their purpose, props, parameters, return values, and behaviour.  Comments on complex logic are insightful and helpful. | Documentation using JSDocs maintained across most milestone applications with occasional inconsistencies.  Most components, functions, hooks, and complex sections have good comments.  Comments generally explain purpose and behaviour well. | Documentation using JSDocs quality varies considerably across milestone applications.  Some components, functions, hooks, and complex sections lack comments.  Existing comments provide basic explanation but may miss important details. | Documentation is inconsistent or missing from large portions of the codebase across multiple milestone applications.  Many components, functions, hooks, and complex sections lack comments.  Existing comments are sparse, unhelpful, or outdated. |
|  | A – 5-4 | B – 3.5 | C – 3-2.5 | D/E – 2-0 |
| Code Formatting (5%) | Exceptional code formatting throughout the codebase. Perfect consistency in indentation, spacing, and use of braces.  Appropriate blank lines between sections for optimal readability.  Proper use of Prettier or similar tool with evidence of automated formatting in all files. Configuration files for formatting tools are properly set up and maintained across all milestone applications. | Good code formatting with minor inconsistencies. Generally consistent indentation, spacing, and use of braces.  Appropriate use of blank lines in most places.  Evidence of automated formatting tool usage with occasional manual overrides or missed files across milestone applications. | Acceptable code formatting but with notable inconsistencies. Basic formatting standards followed but with significant variations.  Inconsistent use of blank lines.  Some evidence of automated formatting tools but applied inconsistently across milestone applications. | Poor code formatting throughout. Inconsistent or improper indentation, spacing, and use of braces.  Minimal use of blank lines or inappropriate placement.  Little to no evidence of automated formatting tools. Code is difficult to read due to formatting issues across most milestone applications. |
|  | A – 5-4 | B – 3.5 | C – 3-2.5 | D/E – 2-0 |
| Dead Code (5%) | No dead code present in the codebase. All files, components, functions, and variables serve a clear purpose.  No commented-out code blocks, unused imports, or redundant declarations.  Code demonstrates exceptional discipline in maintaining only necessary components. | Minimal dead code with minor instances. Few unused elements that do not significantly impact codebase clarity.  Rare instances of commented-out code with clear explanatory notes.  Good discipline in maintaining code cleanliness with occasional oversights across milestone applications. | Some dead code present throughout the codebase. Multiple instances of unused elements that affect codebase clarity.  Several commented-out code blocks without clear explanation.  Basic attention to code cleanliness with notable room for improvement across multiple milestone applications. | Significant dead code throughout the codebase. Many unused or redundant files, components, functions, or variables.  Numerous commented-out code blocks without explanation.  Little attention paid to code cleanliness, creating confusion and potential maintenance issues across most milestone applications. |
|  | A – 10-8 | B – 7.5-6.5 | C – 6-5 | D/E – 4.5-0 |
| Performance and Scalability (10%) | Exceptional attention to performance and scalability. Optimal React patterns and hooks usage.  Efficient state management with proper context or query caching.  Code demonstrates expert understanding of React performance optimisation techniques and scalability principles across all milestone applications. | Good attention to performance and scalability. Efficient React patterns and hooks used in most cases.  Good state management practices.  Code demonstrates good understanding of performance optimisation and scalability principles. | Basic attention to performance and scalability. React patterns and hooks are functional but with room for optimisation.  State management works but lacks optimisation in some cases.  Code demonstrates basic understanding of performance considerations. | Poor attention to performance and scalability. Inefficient React patterns and inappropriate hook usage.  Problematic state management. Excessive re-renders or unnecessary API calls.  Code demonstrates limited understanding of performance optimisation techniques across milestone applications. |

Documentation and Git Usage - Learning Outcome 1 (5%)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A – 5-4 | B – 3.5 | C – 3-2.5 | D/E – 2-0 |
| GitHub Issues and Repository Documentation (2.5%) | Exceptional and consistent use of GitHub issues throughout the entire development process across all milestone applications. Issues are well-organised, properly labelled, and contain detailed descriptions. Progress is clearly tracked with regular updates and appropriate issue closure.  Comprehensive README with all URLs to the applications on Vercel properly documented. Documentation is exceptionally well-organised and easy to follow.  Expert use of Markdown formatting with appropriate headings, lists, tables, code blocks, links, and emphasis. Documentation is visually appealing and highly readable.  Flawless spelling and grammar throughout all documentation. | Good and generally consistent use of GitHub issues throughout development across milestone applications. Issues are organised with appropriate labels and descriptions. Progress is tracked with regular updates.  Good README containing most required URLs with clear documentation. Minor omissions or areas that could be improved in clarity.  Good use of Markdown formatting with appropriate structure. Some advanced formatting features utilised.  Good spelling and grammar with only minor errors that do not impact understanding. | Basic use of GitHub issues but with inconsistent application across milestone applications. Issues have minimal descriptions and limited organisation. Progress tracking is present but sporadic.  Basic README containing some required URLs but with significant omissions or unclear documentation.  Basic use of Markdown with simple formatting. Limited use of advanced features.  Multiple spelling and grammar errors that occasionally impact clarity but do not prevent understanding. | Minimal or inadequate use of GitHub issues. Many issues lack descriptions or organisation. Little to no evidence of progress tracking.  Poor or incomplete README missing multiple required elements. Instructions are unclear, incorrect, or missing entirely.  Poor use of Markdown with little to no formatting. Documentation is difficult to read due to formatting issues.  Numerous spelling and grammar errors that significantly impact clarity and understanding. |
|  | A – 5-4 | B – 3.5 | C – 3-2.5 | D/E – 2-0 |
| Git Configuration and Commit Practices (2.5%) | Perfect implementation of the specified .gitignore file from GitHub. No unnecessary files are tracked, and all appropriate files are ignored. Repository is free from any IDE configurations, node\_modules, build artifacts, or environment files.  Exceptional commit messages that perfectly reflect the context of each functional requirement change. Messages follow a consistent and appropriate naming convention style (e.g., conventional commits). Commit history tells a clear story of project development with logical progression. Commits are appropriately sized with related changes grouped together. | Good implementation of the specified .gitignore file with minor oversights. Few unnecessarily tracked files. Repository is generally clean from system or build files.  Good commit messages that generally reflect functional changes. Messages follow a naming convention with occasional inconsistencies. Commit history shows good progression of development. Commit sizes are generally appropriate. | Basic implementation of the specified .gitignore file with some significant oversights. Some unnecessary files are tracked. Repository contains some files that should be ignored.  Basic commit messages that sometimes reflect functional changes. Inconsistent application of naming conventions. Commit history provides basic understanding of development progression. Commit sizes vary widely with some too large or too small. | Poor implementation of the specified .gitignore file or no custom .gitignore used. Many unnecessary files are tracked. Repository is cluttered with system or build files.  Poor commit messages that rarely reflect functional changes. No consistent naming convention. Commit history does not clearly show development progression. Commits are inappropriately sized (too large or too frequent with minimal changes). |

# ID608001: Intermediate Application Development Concepts

# Project Marking Cover Sheet

Name: Christian Irvine

Date: 10/06/2025

Learner ID: 1000117593

Assessor’s Name: Grayson Orr

Assessor’s Signature: A signature on a white background

AI-generated content may be incorrect.

|  |  |  |
| --- | --- | --- |
| Criteria | Weighting | Mark |
| Functionality | 55 | 55 |
| Code Quality and Best Practices | 40 | 39 |
| Documentation and Git Usage | 5 | 4.5 |
| Final Result | 98.5/100 | |
| This assessment is worth 80% of the final mark for the Intermediate Application Development Concepts course. | | |

Feedback:

**Functionality**  
The application functions as intended. Well done.

**Code Quality and Best Practices**  
Good overall. However, there are commented-out formValues lines in both TodoDisplay.tsx and Quiz.tsx. Additionally, there are unused package.json and package-lock.json files in the root directory. If these files are not needed, consider removing them to keep the project clean.

**Documentation and Git Usage**  
Excellent work overall. The README.md is clear but missing URLs for the Milestone Three and Four applications. Markdown is used reasonably well.