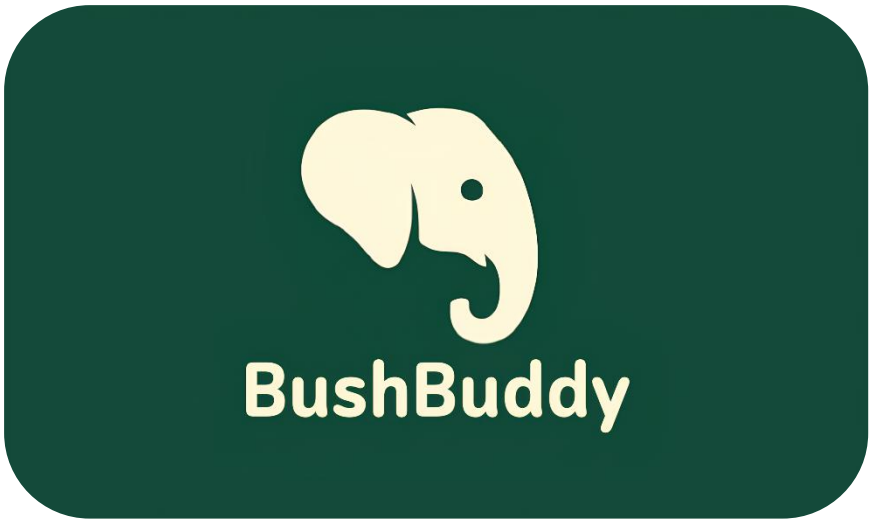
ReturnZero_

20 August 2025

COS301 - Capstone

University of Pretoria



BushBuddy Coding Standards

Version 2

Team members

Ruan Esterhuizen (u23532387)

Ruben Hannes Gadd (u23633353)

Raphael Rato (u22887581)

Tom Schulz (u05039364)

Jean Steyn (u22537229)

Contact

g24capstone@gmail.com

Introduction

This document outlines the coding standards and conventions used in the *AI Powered African Wildlife Detection* project to ensure code uniformity, readability, maintainability, and overall software quality. It includes file structure, naming conventions, formatting guidelines, and tools used for code quality enforcement.

File Structure

```
/root
|
├── src/                          # Main source code
|   ├── backend/                  # Server-side code and application logic
|       ├── api/                  # API route handlers, controllers, and
endpoints
|       └── utils/                # Frontend configuration file (e.g.,
Expo/React Native config)
|
|   └── frontend/                 # Client-side application (e.g., React)
|       ├── app/                  # Core application components and views
|       ├── assets/               # Static files such as images, fonts, and
icons
|       ├── hooks/                # Custom React hooks for shared logic\
|       ├── scripts/              # Utility scripts or setup/initialization
scripts
|       └── app.json              # Frontend configuration file (e.g.,
Expo/React Native config)
|
├── tests/                        # Unit and integration test cases for
backend and frontend
|
├── docs/                         # Project documentation (architecture,
setup guides, etc.)
|
├── config/                       # Configuration files (e.g., environment
variables, service configs)\
|
├── .gitignore                    # Specifies files and folders ignored by
Git version control
└── README.md                     # Project overview, setup instructions,
and usage guide
```

Naming Conventions

- **Files and Folders**: PascalCase for main files and lowercase with hyphens or underscores for folders and not as relevant files (ProfileScreen.js, assets/`)
- **Classes**: PascalCase (e.g., UserProfile, ShapeRenderer)
- **Functions/Methods and Variables** : camelCase (e.g., identifyAnimal, aiResponseData)
- **Constants**: UPPER_CASE (e.g., API_BASE_URL, DEFAULT_TIMEOUT)

Code Formatting Guidelines

- **Indentation**: 2 spaces (no tabs)
- **Line Length**: Max 100 characters
- **Braces**: K&R or Allman style
- **Semicolons**: Always use
- **Quotes**: Use single ' unless string interpolation requires "
- **Spacing**:
    - Use a space after commas and colons.
    - Surround binary operators with a space (e.g., x = y + z)
- **Comments**:
    - Use inline comments sparingly.
    - Use block comments to explain complex logic.
    - Use docstrings for functions and classes.

Best Practices

- Break down UI into reusable, atomic components
- Avoid global variables.
- Use descriptive names.
- Handle errors gracefully using exceptions or error codes.
- Write unit tests for critical components.

AI Detection Integration

- Use RESTful APIs or WebSocket to communicate with the model server
- Send image/text data in base64 or multipart-form format
- Ensure all responses follow a consistent JSON schema: { "success": true, "result": { "animalResult": "Rhino", "confidence": 0.94 } }

Version Control & Collaboration

- Use feature & bugfix branches: feature/detection-ui, bugfix/image-upload
- Follow commit style: type(scope): message
    - Examples: feat(api): add image processing, fix(ui): resolve blurry image issue
- Use Pull Requests (PRs) with peer reviews and CI checks
- Tag releases using vX.Y.Z format

Testing Strategy

- Frontend: Jest, React Testing Library
- Backend: Jest, or Pytest
- Integration tests for AI endpoints using mock model outputs
- Use .env.test and CI scripts for automated tests

Security and Privacy

- Validate and sanitize all user inputs
- Use HTTPS and secure API keys
- Apply rate limiting to detection endpoints

Review Cycle

- Update this document when:
    - Frameworks are upgraded
    - New coding conventions are adopted
    - Major architectural changes are introduced
- Review every quarter (each demo)

Branching Strategy

- **main**: The main production-ready branch.
- **dev**: All features and hotfixes are integrated here before merging into main.,
- **feature/your-feature**: For developing new features.
    - Create a feature/your-feature branch from dev.,
    - You can commit directly to this branch or create sub-branches for specific tasks (e.g., feature/your-feature/subtask).,
- **hotfix/your-hotfix**: For urgent fixes.
    - Create a hotfix/your-hotfix branch directly from main or dev.,
    - You can push directly to this branch or create sub-branches (e.g., hotfix/your-hotfix/subtask).

Workflow

Working in a Feature Branch:

- Create your feature branch from dev (e.g., feature/your-feature).,
- Commit directly to the branch or use sub-branches for more granular tasks.,
- Once complete, submit a pull request to merge your feature branch into dev.,

Working with Hotfixes:

- For urgent fixes, create your hotfix branch from main or dev using the naming convention hotfix/your-hotfix.
- Push changes directly or use sub-branches for specific tasks.
- Open a pull request to merge your hotfix into both dev and main.

Branch Rulesets

- **main:**
    - No direct pushes allowed; all changes must go through pull requests.
        - **Requires at least 3 approvals** before merging.,
        - Must maintain a linear history (no merge commits).,
        - All commits must be signed and verified.,
- **dev:**
    - No direct pushes allowed; all changes must go through pull requests.,
        - **Requires at least 2 approvals** before merging.,
        - Must maintain a linear history (no merge commits).
- **feature/***:
    - Direct commits are allowed, or you can opt to create sub-branches for specific tasks.,
        - Once your feature is complete, a pull request is required to merge into dev
- **hotfix/***:
    - Direct pushes are allowed, or you can opt to create sub-branches for specific tasks.,
        - A pull request is required to merge into both dev and main.