Arya Putatunda
arya6v9000@gmail.com Kolkata, India
Timezone: Indian Standard Time
(UTC+5:30)

Linter for Concerto

GSoC Project proposal for Accord Project

PERSONAL DETAILS & CONTACT INFORMATION :

- **Full Name**: Arya Putatunda
- **University:** SRM Institute of Science and Technology Delhi NCR Ghaziabad
- **Date of Enrollment:** September 2022
- **Expected Graduation date:** May 2026
- **Degree:** Bachelors of Technology
- **Email Address**: arya6v9000@gmail.com
  ap9857@srmist.edu.in
- **Github Username**: https://github.com/Movazed
- **Phone Number**: 8981938824

TABLE OF CONTENTS

SYNOPSIS

The Concerto modeling language is used in the Accord Project for defining domain-specific schemas and smart legal contracts. However, ensuring code consistency, best practices, and error prevention currently relies on manual review.
This project proposes a TypeScript-based linter for Concerto that:

- Automates validation of naming conventions, decorators, and structural rules.
- Prevents anti-patterns (e.g., unsafe regex, missing validators).
- Enforces project-specific standards (e.g., @Term decorators, namespace hierarchies).

Impact: Faster development, fewer bugs, and better adherence to Accord Project guidelines.

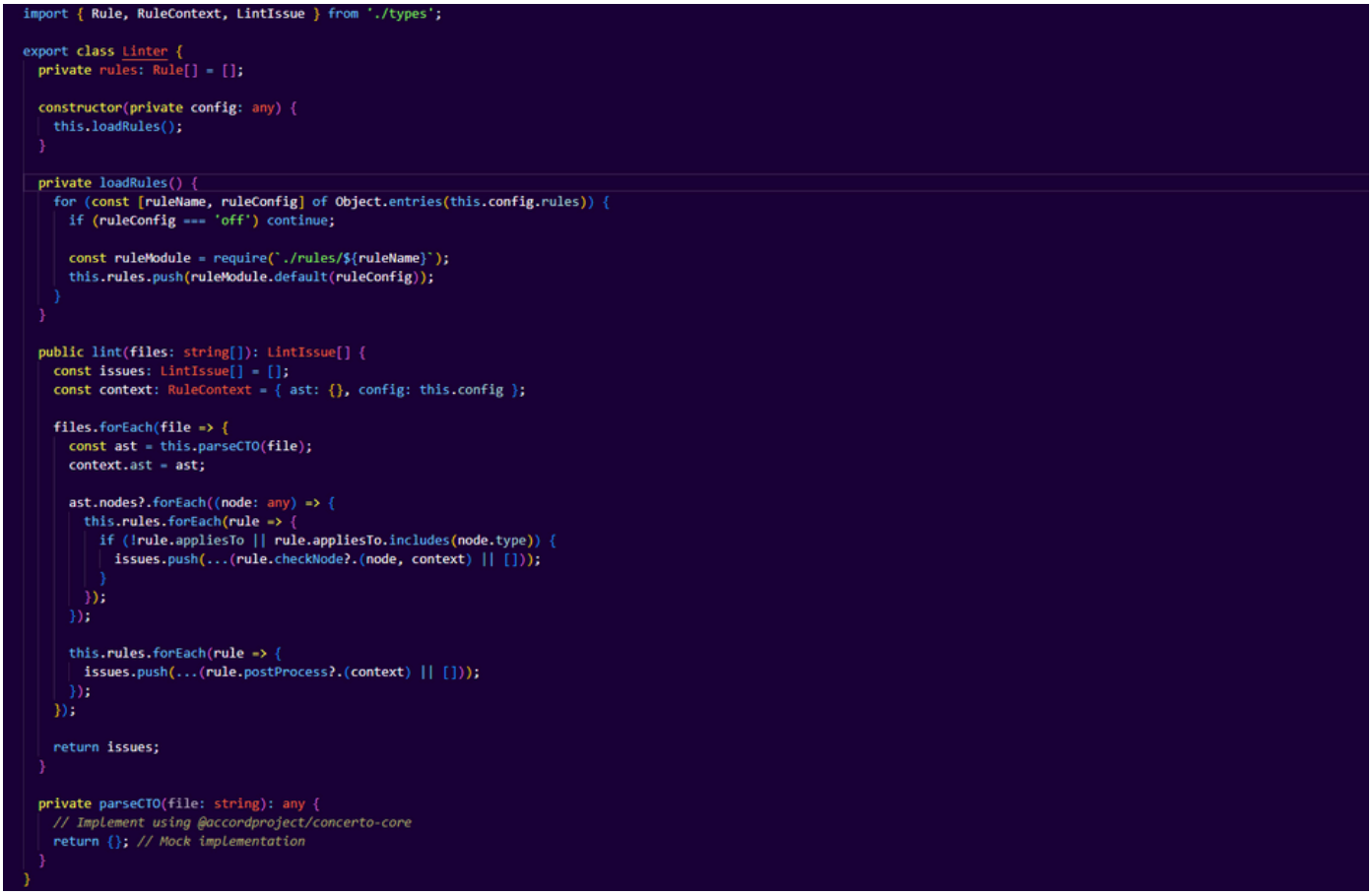
BENEFITS TO THE COMMUNITY

The purpose of this project is to create a linter for, **Accord Project**. The linter of this component will be -

- Consistency: Ensures all .cto files follow naming and structural conventions.
- Accessibility: Catches accessibility gaps (e.g., missing aria-labels in generated UIs).
- Extensibility: Teams can add custom rules via a config file (.concerto-lint.json).
- Integration: Works with IDEs and CI/CD pipelines (e.g., GitHub Actions

EXPECTED RESULTS

This is a sample code representation of the expected final, ready-to-use component.



```
import { Rule, RuleContext, LintIssue } from './types';

export class Linter {
  private rules: Rule[] = [];

  constructor(private config: any) {
    this.loadRules();
  }

  private loadRules() {
    for (const [ruleName, ruleConfig] of Object.entries(this.config.rules)) {
      if (ruleConfig === 'off') continue;

      const ruleModule = require(`./rules/${ruleName}`);
      this.rules.push(ruleModule.default(ruleConfig));
    }
  }

  public lint(files: string[]): LintIssue[] {
    const issues: LintIssue[] = [];
    const context: RuleContext = { ast: {}, config: this.config };

    files.forEach(file => {
      const ast = this.parseCTO(file);
      context.ast = ast;

      ast.nodes?.forEach((node: any) => {
        this.rules.forEach(rule => {
          if (!rule.appliesTo || rule.appliesTo.includes(node.type)) {
            issues.push(...(rule.checkNode?.(node, context) || []));
          }
        });
      });

      this.rules.forEach(rule => {
        issues.push(...(rule.postProcess?.(context) || []));
      });
    });

    return issues;
  }

  private parseCTO(file: string): any {
    // Implement using @accordproject/concerto-core
    return {}; // Mock implementation
  }
}
```

where,
Linter.ts - Linter for the Concerto ASI module.

cli.ts ready-to-use CLI tool.

NOTE*: There might be more components between the Scripts and the JSON, APIs which are not shown above*

Representation of Rules in Linter

```json
{
  "rules": {
    "scalar-camel-case": "error",
    "require-term-decorator": { "level": "warning" },
    "no-maps": "error",
    "extends-base-concept": {
      "level": "error",
      "options": { "namespace": "org.example", "baseConcept": "Base" }
    }
  }
}
```

Sample of Config.json

```typescript
#!/usr/bin/env node
import { LintIssue } from './types';
import { program } from 'commander';
import { Linter } from './linter';
import * as path from 'path';
import * as fs from 'fs';

program
  .version('1.0.0')
  .argument('<files...>', 'Concerto files to lint')
  .option('-c, --config <path>', 'Path to config file', './config.json')
  .action((files, options) => {
    const config = loadConfig(options.config);
    const linter = new Linter(config);
    const results = linter.lint(files);
    console.log(formatResults(results));
  });

program.parse();

function loadConfig(configPath: string): any {
  const absolutePath = path.resolve(process.cwd(), configPath);
  return JSON.parse(fs.readFileSync(absolutePath, 'utf-8'));
}

function formatResults(results: LintIssue[]): string {
  return results.map(issue =>
    `${issue.level.toUpperCase()}: ${issue.message}` +
    (issue.location ? ` (Line ${issue.location.line})` : '')
  ).join('\n');
}
```

cli.ts code of in typescript

APPROACH

The linter adopts a modular, AST-based architecture that separates parsing, rule validation, and reporting into distinct layers for maximum flexibility and maintainability.

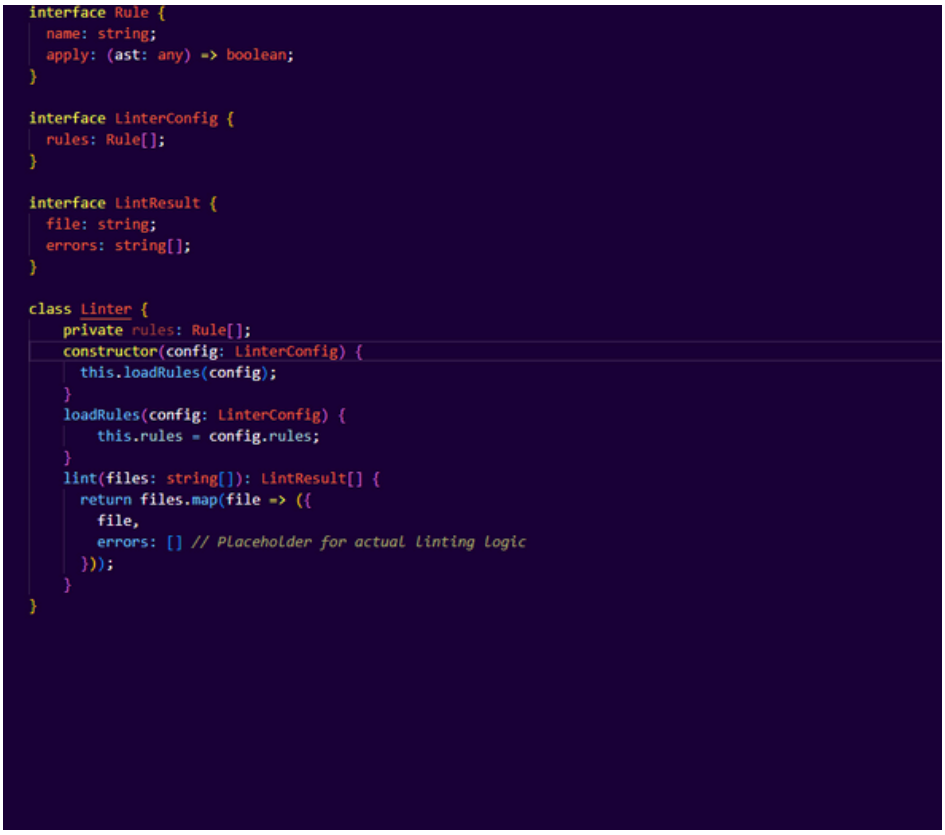
PHASE I - CORE LINTER ENGINE

STEP 1 :

1. AST Parsing: Use @accordproject/concerto-core to parse .cto files.
2. Rule System: Define rules in TypeScript (e.g., checkCamelCase, validateTermDecorator).
3. CLI Tool: Support --fix to auto-correct issues like naming conventions.

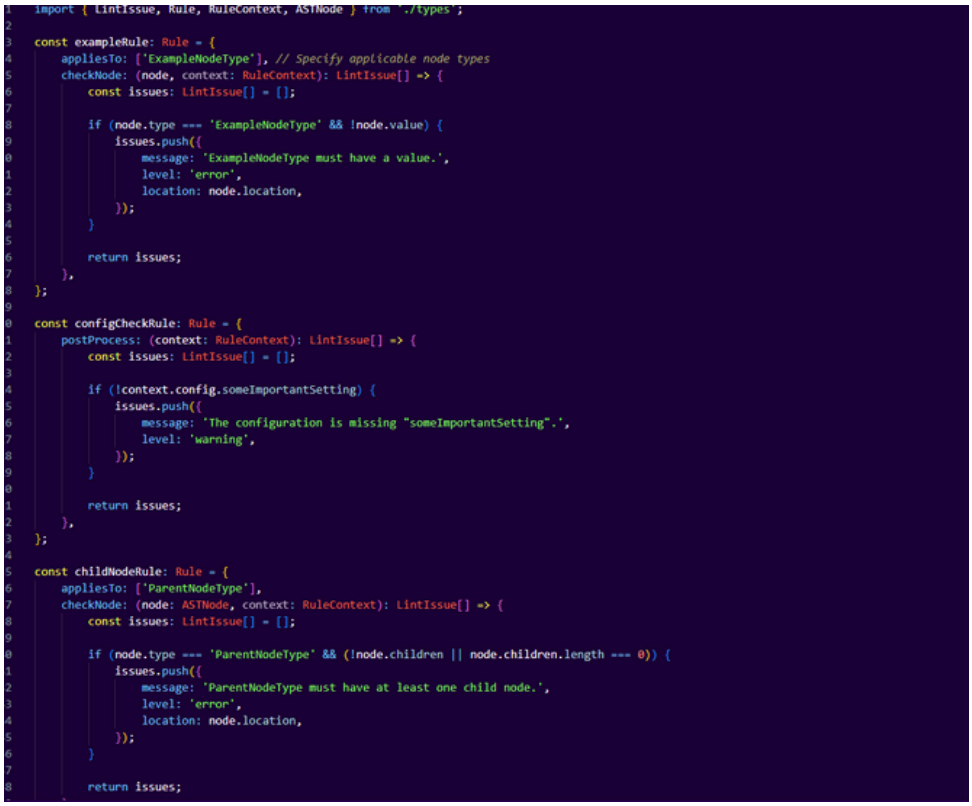
Phased Implementation:

- Set up AST parsing infrastructure using @accordproject/concerto-core



```
interface Rule {
  name: string;
  apply: (ast: any) => boolean;
}

interface LinterConfig {
  rules: Rule[];
}

interface LintResult {
  file: string;
  errors: string[];
}

class Linter {
    private rules: Rule[];
    constructor(config: LinterConfig) {
      this.loadRules(config);
    }
    loadRules(config: LinterConfig) {
        this.rules = config.rules;
    }
    lint(files: string[]): LintResult[] {
      return files.map(file => ({
        file,
        errors: [] // Placeholder for actual linting logic
      }));
    }
}
```

- Develop core rule interface:



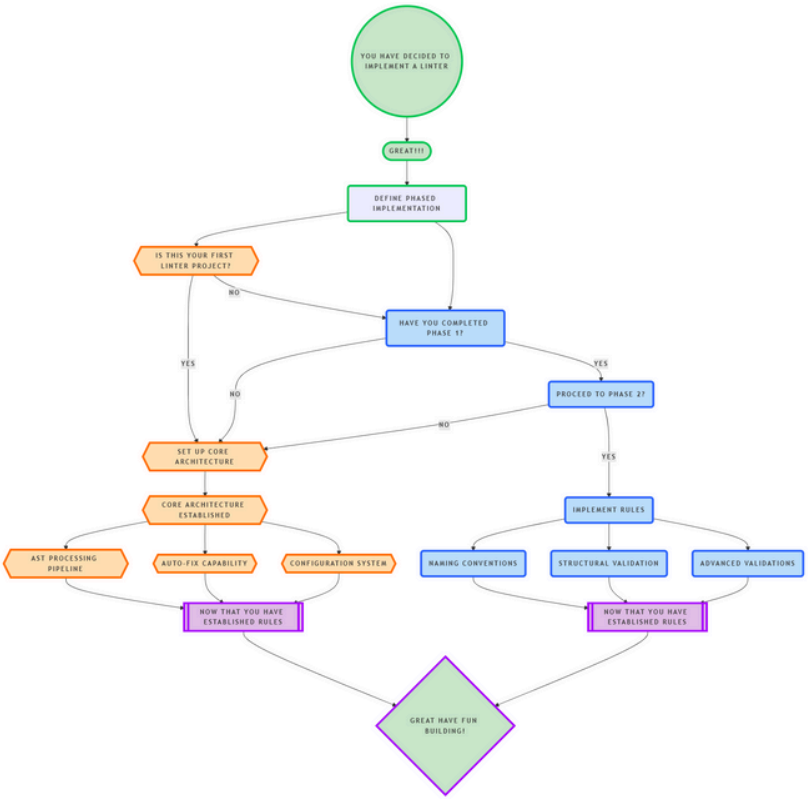
```
import { LintIssue, Rule, RuleContext, ASTNode } from './types';

const exampleRule: Rule = {
    appliesTo: ['ExampleNodeType'], // Specify applicable node types
    checkNode: (node, context: RuleContext): LintIssue[] => {
        const issues: LintIssue[] = [];

        if (node.type === 'ExampleNodeType' && !node.value) {
            issues.push({
                message: 'ExampleNodeType must have a value.',
                level: 'error',
                location: node.location,
            });
        }

        return issues;
    },
};

const configCheckRule: Rule = {
    postProcess: (context: RuleContext): LintIssue[] => {
        const issues: LintIssue[] = [];

        if (!context.config.someImportantSetting) {
            issues.push({
                message: 'The configuration is missing "someImportantSetting".',
                level: 'warning',
            });
        }

        return issues;
    },
};

const childNodeRule: Rule = {
    appliesTo: ['ParentNodeType'],
    checkNode: (node: ASTNode, context: RuleContext): LintIssue[] => {
        const issues: LintIssue[] = [];

        if (node.type === 'ParentNodeType' && (!node.children || node.children.length === 0)) {
            issues.push({
                message: 'ParentNodeType must have at least one child node.',
                level: 'error',
                location: node.location,
            });
        }

        return issues;
```

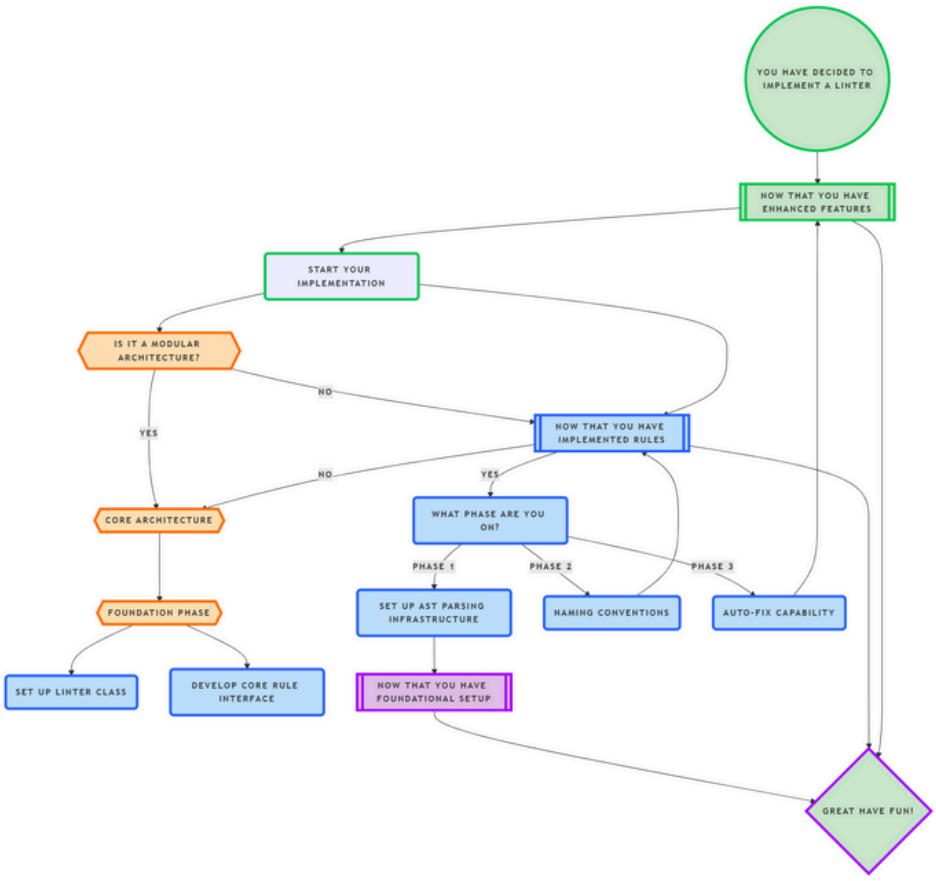
- Develop rule configurations:

```
import { RuleFactory } from './types';

export const scalarCamelCaseRule: RuleFactory = (options) => ({
  appliesTo: ['ScalarDeclaration'],
  checkNode: (node) => {
    if (!/^[a-z][a-zA-Z0-9]*$/.test(node.name)) {
      return [{
        message: `Scalar "${node.name}" must be camelCase.`,
        level: options.level || 'error',
        location: node.location,
      }];
    }
    return [];
  },
});

export const requireTermDecoratorRule: RuleFactory = (options) => ({
  appliesTo: ['*'],
  checkNode: (node) => {
    if (!node.decorators?.some((d: any) => d.name === 'Term')) {
      return [{
        message: `Missing @Term decorator on ${node.type} "${node.name}"`,
        level: options.level || 'warning',
        location: node.location,
      }];
    }
    return [];
  },
});
```

STEP 2:

- Develop rule configurations:





STEP 3:

1. **Rule Registration:**
   - Load accessibility rules from the project's .concerto-lint.json config.
   - Enable/disable rules based on preset profiles (base/strict).
2. **AST Traversal:**
   - Scan table declarations for missing accessibility metadata.
   - Verify responsive attributes in wide or complex layouts.
3. **Reporting:**
   - Classify issues by severity (error/warning/suggestion).
   - Output machine-readable reports (JSON) for CI/CD pipelines.

PHASE 2 - RULE IMPLEMENTATION

STEP 4:

❖ createNamingRule({
pattern: /^[a-z][a-zA-Z0-9]*$/, // camelCase
types: ['Property','Scalar']});

STEP 5:

❖ createHierarchyRule({
baseConcept: 'BaseEntity',
requiredDecorators: ['@Term']});

STEP 6:

❖ Advanced Validations:

```typescript
export interface LintIssue {
    message: string;
    level: 'error' | 'warning';
    location?: {
      line: number;
      column: number;
    };
  }

  export interface RuleContext {
    ast: any; // Concerto AST
    config: any;
  }
  export interface ASTNode {
    type: string;
    value?: string;  // Optional value property
    children?: ASTNode[]; // Optional children for tree structure
    location?: {
        line: number;
        column: number;
    };
}
  export interface Rule {
    appliesTo?: string[];
    checkNode?: (node: any, context: RuleContext) => LintIssue[];
    postProcess?: (context: RuleContext) => LintIssue[];
  }

  export type RuleFactory = (options: any) => Rule;
```

- Regex validator safety checks
- Type reference validation
- Circular dependency detection

EXTENDED GOALS :

```typescript
import { LintIssue, Rule, RuleContext, ASTNode } from './types';

const exampleRule: Rule = {
    appliesTo: ['ExampleNodeType'], // Specify applicable node types
    checkNode: (node, context: RuleContext): LintIssue[] => {
        const issues: LintIssue[] = [];

        if (node.type === 'ExampleNodeType' && !node.value) {
            issues.push({
                message: 'ExampleNodeType must have a value.',
                level: 'error',
                location: node.location,
            });
        }

        return issues;
    },
};

const configCheckRule: Rule = {
    postProcess: (context: RuleContext): LintIssue[] => {
        const issues: LintIssue[] = [];

        if (!context.config.someImportantSetting) {
            issues.push({
                message: 'The configuration is missing "someImportantSetting".',
                level: 'warning',
            });
        }

        return issues;
    },
};

const childNodeRule: Rule = {
    appliesTo: ['ParentNodeType'],
    checkNode: (node: ASTNode, context: RuleContext): LintIssue[] => {
        const issues: LintIssue[] = [];

        if (node.type === 'ParentNodeType' && (!node.children || node.children.length === 0)) {
            issues.push({
                message: 'ParentNodeType must have at least one child node.',
                level: 'error',
                location: node.location,
            });
        }

        return issues;
```

interface FixableRule extends Rule {provideFixes(node: ASTNode): Fix[];}

{"rules": {"naming/camel-case": ["error", {"properties": true,"concepts": false}],"validation/require-validator": "warn"}}

TIMELINE & DELIVERABLES

	PERIOD	TASK
MILESTONE 1	After proposal submission [April 9 - May 7]	**Objectives:** Refine technical design with mentors Set up GitHub repo with CI/CD Conduct preliminary research on Concerto AST **Outcome:** Draft implementation plan in DESIGN.md
	Week 1 and 2 [May 8 - June 1]	**Key Activities:** Onboard with Accord Project tools Finalize rule priority matrix Prepare test suite scaffolding **Deliverable**: Public project roadmap Initial prototype (proof-of-concept)
	Week 3 and 4 [June 2 - August 25]	Phase 1: Core Linter (June 2 - July 14) Week Focus Area Deliverable 1-2 AST Parser + CLI Basic file scanning 3-4 Rule Engine Configurable rules system 5-6 Core Validations 10+ essential rules
MILESTONE 2	Week 5 and 6 [June 26 - July 8]	Submit: Working CLI tool (v0.3.0) Documentation scaffold 50+ passing tests
	Week 7 and 8 [July 14 - July 18]	VS Code extension skeleton
	Week 9 and 10 [July 14 - Aug 18]	Production-ready v1.0.0 Demo video (5 min) Contributor guide
	Week 11 and 12 [Aug 25 - Sep 1]	Extended Goals (if applicable): React/Vue codegen integration Enterprise feature additions

ADDITIONAL INFORMATION

About Me :

Hey! This is Arya Putatunda (or Movazed) , a pre-final yearite from the SRM IST Delhi NCR Ghaziabad, India. I am a full stack web developer proficient in React, Tailwind CSS, JavaScript & Next Js along with experience in working with Angular, ML & CI/CD Pipelines. I love exploring new technologies & contributing to the Open-Source community, making a positive difference with my code.

Skills :

Skill Name	Proficiency (1-5)
C++	4
Tailwind CSS	4.5
Python	4
Git/Github	4.5
ReactJS/NextJS/Angular	3.5

Why Learning Equality ?

The equity-driven values. The idea that drives Learning Equality. The work culture, the sections of the community they target, and the entire ideology that runs this organisation.

My idea of technology has always been to put it to use where it can positively impact the lives of people who really need it. And Learning Equality truly aligns with all of it. Throughout these few months that I have worked with Learning Equality, it has been one of my most cherishable Open-Source contribution experiences so far. The entire community, from maintainers to fellow contributors, is all so welcoming and willing to help. It makes me happy to be part of a team that shares equal enthusiasm and motivation towards creating a more equitable society as me :)

Plans after GSoC :

1. Continued Contribution
2. Regardless of GSoC outcomes, I intend to:
    - Maintain and enhance the Concerto Linter as part of Accord Project
    - Mentor new contributors through documentation and code reviews
    - Explore integration with Kolibri's educational tools
3. Long-Term Vision
    - Bridge the digital divide by improving accessible tech solutions
    - Adapt this linter for use in low-bandwidth learning environments

Other Commitments:

I do not have any other plans and commitments this summer except GSoC. I'll have my mid-semester exams somewhere in the month of July, the dates of which have not been announced yet. However, I have allotted a week of buffer period at half the GSoC coding period to meet these commitments.

THANK YOU!