**SIMATS SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**CHENNAI-602105**

**Generating Code From Natural Language Descriptions**

**A CAPSTONE PROJECT REPORT**

*Submitted in the partial fulfillment for the award of the degree of*

**BACHELOR OF ENGINEERING**

**IN**

**INFORMATION TECHNOLOGY**

**Submitted by**

**VASALA NANDA GOPALA KRISHNA (192211707)**

**MAHESH CHOWDARY (192211116)**

**SUKUMAR.R (192211346)**

**Under the Supervision of**

**Dr. G. Michael**

**FEBRUARY 2024**

**DECLARATION**

We, **Vasala Nanda Gopala Krishna, Mahesh Chowdary, Sukumar. R,** students of **‘Bachelor of Engineering in Information Technology**, Department of Computer Science and Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the work presented in this Capstone Project Work entitled **Automated Network Security Testing Tools** is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics.

(Vasala Nanda Gopala Krishna 192211707)

(Mahesh Chowdary 192211116)

(Sukumar. R 192211346)

Date:

Place:

**CERTIFICATE**

This is to certify that the project entitled **“Generating Code From Natural Language Descriptions”** submitted by **Vasala Nanda Gopala Krishna, Mahesh Chowdary, Sukumar. R** has been carried out under our supervision. The project has been submitted as per the requirements in the current semester of B. Tech Information Technology.

Teacher-in-charge

Dr. G.Michael

**Table of Contents**

|  |  |
| --- | --- |
| **S.NO** | **TOPICS** |
| 1 | **Abstract** |
| 2 | **Introduction** |
| 3 | **Problem Statement** |
| 4 | **Proposed Design**   1. Requirement Gathering and Analysis 2. Tool selection criteria 3. Scanning and Testing Methodologies |
| 5. | **Functionality**   1. User Authentication and Role Based Access Control. 2. Tool Inventory and Management 3. Security and Compliance Control |
| 6 | **UI Design**   1. Layout Design 2. Feasible Elements Used 3. Elements Positioning and Functionality |
| 7 | **Conclusion** |

**ABSTRACT:**

**Abstract:**

Generating code from natural language descriptions is a promising area of research that aims to bridge the gap between human-readable text and executable software code. This paper explores various approaches and techniques for automatically translating natural language specifications into code, enabling developers to express their intentions in familiar language constructs and seamlessly generate corresponding code snippets or scripts. We delve into the challenges and opportunities associated with this task, including natural language understanding, semantic parsing, code generation, and evaluation metrics. Furthermore, we review existing methodologies, tools, and frameworks for code generation from natural language descriptions, highlighting their strengths, limitations, and potential applications in software development. Through this exploration, we aim to provide insights into the state-of-the-art techniques and advancements in this field, along with future directions and opportunities for research and development.

**Introduction:**

Generating code from natural language descriptions has emerged as a transformative approach to software development, aiming to bridge the gap between human language and executable code. Traditionally, writing code has been the domain of skilled programmers who possess proficiency in programming languages and syntax. However, this barrier to entry can be daunting for individuals with limited programming experience or domain expertise.

In response to this challenge, researchers and practitioners have explored methods to automate the process of code generation from natural language descriptions. By enabling developers to express their intentions and requirements in familiar language constructs, this approach promises to democratize software development, empower non-programmers to contribute to the creation of software systems, and accelerate the development cycle.

**Problem Statement:**

The process of translating natural language descriptions into executable code poses significant challenges in software development, particularly for individuals with limited programming experience or domain expertise. Traditional programming requires proficiency in specific programming languages and syntax, creating a barrier to entry for non-programmers and hindering collaboration between developers and domain experts.

**Proposed Design:**

**Requirements Gathering and Analysis:** Conduct stakeholder interviews and surveys to learn about the organization's network infrastructure, security objectives, and specific testing requirements.

**Stakeholder Identification:** Identify stakeholders involved in the code generation process, including developers, domain experts, end-users, and project managers.

**Functional Requirements**: Develop algorithms and models for parsing and understanding natural language descriptions, including syntax, semantics, and intent recognition.

**Functionality:**

**Natural Language Parsing:**

* Implement algorithms and techniques to parse natural language descriptions provided by users.
* Extract relevant keywords, entities, and relationships from the natural language input

**Tool Inventory and Management:**

* Utilize NLP libraries such as NLTK (Natural Language Toolkit), spaCy, or StanfordNLP to perform tokenization, part-of-speech tagging, and dependency parsing of natural language descriptions.
* Leverage machine learning frameworks like TensorFlow or PyTorch to train models for semantic understanding and intent recognition.

**Security and Compliance Controls:**

* Implement access control mechanisms to restrict access to sensitive data and code generation tools.

**Architectural Design:**

**Presentation Layer:**

* A web-based user interface (UI) for engaging with the assessment framework.
* Use role-based access control (RBAC) to handle user authentication and permissions.

**Application Layer:**

* Develop NLP modules to parse and analyze natural language descriptions provided by users.
* Utilize NLP techniques such as tokenization, part-of-speech tagging, and syntactic parsing to extract relevant information from the input text.

**Monitoring and Management Layer:**

* Implement monitoring capabilities to track the performance of the code generation system, including response times, resource utilization, and throughput.
* Monitor key performance indicators (KPIs) such as processing speed, memory usage, and system stability to ensure optimal performance and scalability.

**UI Design:**

**Dashboard:**

Provides a summary of the assessment framework, including the number of current tests, most recent test results, and system status indicators**.**

**User Management:**

* Implement user authentication mechanisms to verify the identity of users accessing the code generation system.
* Allow users to customize their preferences, such as preferred programming languages, code formatting options, and default templates for generated code.

**Help and Support:**

* Links to user manuals, tutorials, and documentation materials for understanding how to utilize the assessment framework efficiently.
* Contact details for technical help, FAQs, and community forums for asking questions and sharing best practices.

**Feasible Element Used:**

**Dashboard:**

Provide an overview section that summarizes key metrics and statistics related to code generation activities and also display metrics such as total number of code generation requests, average processing time, and distribution of requests by user or project.

**User Management:**

* Implement user authentication mechanisms to verify the identity of users accessing the code generation system.
* Define role-based access control (RBAC) policies to assign permissions and privileges to users based on their roles and responsibilities within the organization.

**Help and Support:**

* Positioned on the dashboard to provide real-time monitoring of network security.
* Widgets offer live statistics such as active scans, discovered threats, and system resource utilization.

**Element Positioning and Functionality:**

**Real-time Monitoring:**

* Positioned on the dashboard to provide real-time monitoring of network security.
* Widgets offer live statistics such as active scans, discovered threats, and system resource utilization.

**Collaboration Features:**

* Enable real-time collaborative editing, allowing multiple users to work on the same code generation task simultaneously.
* Allow users to add comments and annotations to code snippets or natural language descriptions, facilitating communication and feedback among collaborators.

**Trend Analysis:**

* Analyze the adoption rate of code generation from natural language descriptions over time, tracking the number of users or organizations utilizing the technology.
* Monitor trends in programming language preferences for generating code from natural language descriptions.

**Conclusion:**

Generating code from natural language descriptions represents a significant advancement in software development, offering a promising approach to bridging the gap between human-readable text and executable code. Through the automation of the code generation process, this technology aims to democratize software development, empower non-programmers to contribute to the creation of software systems, and accelerate the development cycle**.**