# TumblAI Project Report

(By Gerard Ledoux)

**Version 1.0**

## 1. Introduction

### Overview:

TumblAI is an innovative Chrome extension designed to enhance user engagement on Tumblr by leveraging the power of AI. This extension enables users to generate AI-powered comments on Tumblr posts, tailored to various tones such as friendly, funny, and disagreeable. TumblAI aims to simplify the interaction process on social media, making it more dynamic and engaging.

## 2. Objectives

### Primary Objectives:

- To create a minimal and user-friendly Chrome extension.
- To enable users to generate AI-powered comments with a selectable tone.
- To seamlessly integrate the comment generation functionality into the Tumblr platform.

### Secondary Objectives:

- To ensure the extension is easily toggleable via a user interface.
- To manage backend services for generating comments using predefined responses and AI models.
- To maintain smooth and responsive performance without interrupting the user's browsing experience.

## 3. System Design

### Architecture:

The TumblAI project is built using a combination of React for the frontend and Python (Flask) for the backend. The architecture can be divided into three main components:

- **Frontend (Chrome Extension)**
- **Backend (Flask API)**
- **Integration with Tumblr**

### Components:

1. **Frontend (Chrome Extension)**

- **React Components**: Utilized for building the user interface, including the popup window and comment generation interface.
- **Switch Component**: Allows users to toggle the extension on and off.
- **Popup Window**: Provides the main interface for interacting with the extension.

1. **Content Script**: Injects the comment box into the Tumblr page and handles interactions with the Tumblr DOM.

2. **Backend (Flask API)**
    2. **Flask Server**: Handles incoming requests from the extension and returns AI-generated comments based on the selected tone.
    3. **Predefined Responses**: For initial testing, the backend returns predefined responses stored in a JSON file.
    4. **AI Model Integration**: Future iterations will integrate more sophisticated AI models to generate tailored comments such as openai, haiku or Free-Auto-GPT.

3. **Integration with Tumblr**
    5. **DOM Manipulation**: Uses JavaScript to manipulate the Tumblr DOM, adding the comment box and handling user interactions.
    6. **Event Listeners**: Listens for specific events such as button clicks and URL changes to ensure the extension functions correctly across different pages and states.

# 4. Implementation: Frontend

## 4.1 PopUpWindow.jsx

The **PopUpWindow** component initializes the extension and provides the toggle functionality to enable or disable the extension.

```
/* eslint-disable no-undef */
import React, { useState } from 'react'
import './PopUpWindow.scss'
import Switch from './Switch.jsx'

const PopUpWindow = () => {
    /* Prototype to turn extension on and off */
    let [ isToggled, setIsToggled ] = useState(true);
    const handleToggle = () => {
            setIsToggled(!isToggled);
            if (!isToggled) {
            // Send a message to the content script to enable functionality
            chrome.tabs.query({ active: true, currentWindow: true }, (tabs) => {
                chrome.tabs.sendMessage(tabs[0].id, { action: 'ENABLE_EXTENSION'
});
```

```
            });
            } else {
            // Send a message to the content script to disable functionality
            chrome.tabs.query({ active: true, currentWindow: true }, (tabs) => {
                chrome.tabs.sendMessage(tabs[0].id, { action: 'DISABLE_EXTENSION'
});
            });
            }
    };

    return (
    <div className='pop-up-window'>
        <div className='title'>
            <span className='tumblr'>TUMBLR</span>
            <span className='ai'>AI</span>
        </div>
        <div className='onOff'>
            <span className='label'>ON/OFF</span>
            <Switch isToggled={isToggled} onToggle={handleToggle}/>
        </div>
    </div>
    )
}

export default PopUpWindow
```

## 4.2 Content.js

The **Content** component manages the insertion of the comment box into the Tumblr page and handles user interactions.

```
/* eslint-disable no-undef */
import React from 'react';
import ReactDOM from 'react-dom/client';
import CommentComponent from './components/CommentComponent.jsx';

let extensionEnabled = true;
let observer;
const observerOptions = {
    childList: true,
    subtree: true,
};

function addCommentBoxToReplySection(post, replyButton) {
```

```
    // Logic to close previously created comment box when another reply section
is opened
    // let isCommentBoxPresent = document.getElementById("commentBox");
    // if (isCommentBoxPresent) {
    //     console.log("Removing previous comment box");
    //     isCommentBoxPresent.remove();
    // }

    // const replySectionObserver = new MutationObserver((mutationsList,
observer) => {
    //     for (let mutation of mutationsList) {
    //       if (mutation.type === 'childList' && mutation.addedNodes.length > 0)
{
    //         mutation.addedNodes.forEach(node =&gt; {
    //           if (node.nodeType === 1 && node.classList.contains('EnRJg')) {
    //             replySectionObserver.disconnect(); // Stop observing once the
reply section is found
    //             executeAfterReplyLoaded(post, replyButton);
    //           }
    //         });
    //       }
    //     }
    // });

    // Not the best way down here but it works. Use time benchmarks to adjust
timing or
    // Implement function above (not completed yet)

    setTimeout ( () => {
        const replySection = document.getElementsByClassName("EnRJg")[0];
        if (replySection) {
            const postInfo = {
                userName: post.getElementsByClassName("W9hfZ")[0] ?
post.getElementsByClassName("W9hfZ")[0].innerText : null,
                description: post.getElementsByClassName("LaNUG")[0] ?
post.getElementsByClassName("LaNUG")[0].innerText : null,
                tags: post.getElementsByClassName("hAFp3")[0] ?
post.getElementsByClassName("hAFp3")[0].innerText : null
            }

            const commentBox = document.createElement('div');
            commentBox.id = 'commentBox';

            replySection.insertBefore(commentBox,
document.getElementsByClassName("rEGcu tprzO fYhK7")[0]);
```

```
            const root = ReactDOM.createRoot(commentBox)
            root.render(<CommentComponent replyButton={replyButton}
postInfo={postInfo}/>);
        }
    }, 500);

}

function addListener(post) {
    const buttons = post.getElementsByClassName("TRX6J");

    Array.from(buttons).forEach(isReplyButton => {
        if (isReplyButton.ariaLabel === 'Reply') {
            if (!isReplyButton.dataset.listenerAttached) {
                isReplyButton.dataset.listenerAttached = true;
                isReplyButton.addEventListener('click', () =>
addCommentBoxToReplySection(post, isReplyButton));
            }
        }
    });

}

const addListenerToReplyButton = async (newPosts, observer) => {
    for (let post of newPosts) {
        // Check if new node is a new post
        if (post.type === 'childList' &&
            (post.target.className === "rZlUD KYCZY W45iW" ||
             post.target.className === "rZlUD KYCZY F4Tcn" ||
             post.target.className === "rZlUD F4Tcn" ||
             post.target.className === "rZlUD W45iW"
            )
        )
        {
            addListener(post.target);
        }
    }
};

function handleExistingPosts(postsContainer) {
    let initialPosts = postsContainer.getElementsByClassName("rZlUD");
    Array.from(initialPosts).forEach(post => {
        addListener(post);
    });
}
```

```
function initializeObserver(retries = 10, delay = 200) {
    let targetNode = document.getElementsByClassName("zAlrA")[0];
    if (targetNode && targetNode.children.length > 2) {
        if (observer) {
            observer.disconnect();
        }
        observer = new MutationObserver(addListenerToReplyButton);
        observer.observe(targetNode, observerOptions);
        handleExistingPosts(targetNode);
    } else if (retries > 0) {
        setTimeout(() => initializeObserver(retries - 1, delay), delay);
    } else {
        console.error("Failed to find the target node.");
    }
}

/* Prototype to turn extension on and off */
chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
  if (message.action === 'ENABLE_EXTENSION') {
    extensionEnabled = true;
    console.log("Extension is enabled");
    // initializeObserver();  // Re-initialize the observer when the extension is
enabled
  } else if (message.action === 'DISABLE_EXTENSION') {
    extensionEnabled = false;
    console.log("Extension is disabled");
    if (observer) {
    //   observer.disconnect();  // Disconnect the observer when the extension is
disabled
    }
  }
});

chrome.runtime.onMessage.addListener((message, sender, response) => {
    if (message.type === "NEW_PAGE_LOAD" || message.type === "URL_CHANGED") {
        initializeObserver();
    }
});
```

# 5. Implementation: Backend

## 5.1 SampleAPI.py

A simple Flask API to handle incoming requests and return AI-generated comments.

```
from flask import Flask, request, jsonify
```

```python
from flask_cors import CORS
import json

app = Flask(__name__)
CORS(app, resources={r"/*": {"origins": "*"}})

with open('TestComments.json', 'r') as f:
    comments = json.load(f)

@app.route('/tumblrAI', methods=['POST', 'OPTIONS'])
def get_comment():
    if request.method == 'OPTIONS':
        response = app.make_default_options_response()
        headers = request.headers.get('Access-Control-Request-Headers', '')
        response.headers.add("Access-Control-Allow-Headers", headers)
        response.headers.add("Access-Control-Allow-Methods", "POST, OPTIONS")
        return response

    data = request.get_json()

    if not data or 'selectedTone' not in data:
        return jsonify({"error": "Invalid request"}), 400

    selected_tone = data['selectedTone'].lower()

    if selected_tone in comments:
        response = {
            "comment": comments[selected_tone]
        }
        return jsonify(response)
    else:
        return jsonify({"error": "Tone not found"}), 404

if __name__ == '__main__':
    app.run(host="localhost", port=5000, debug=True)
```

## 5.2 TestComments.json

A JSON file with predefined comments for different tones.

```json
{
    "friendly": "This is a friendly comment",
    "funny": "This is a funny comment",
    "disagree": "I disagree with this"
}
```

# 6. Testing

## 6.1 Unit Testing

Unit testing was implemented to ensure that individual components of the TumblAI project work as expected. While deep unit tests were not implemented, basic tests were created to verify that key components are loading and rendering correctly. The testing focused on ensuring that all requirements listed in the Kanban board were met, serving as a form of validation for the project's functionality.

### Popup Component Testing

A simple unit test was created to check if the popup component loads and contains the specific content necessary for the extension's functionality.

```
import { render, screen } from '@testing-library/react';
import PopUpWindow from './components/PopUpWindow';

test('renders learn react link', () => {
  render(<PopUpWindow />);
  const linkElement = screen.getByText(/ON/i);
  expect(linkElement).toBeInTheDocument();
});
```

This test ensures that the popup window renders and contains the "ON/OFF" label, indicating that the component is loading as expected.

## 6.2 Integration Testing

Integration testing was conducted to verify that the frontend and backend components interact correctly. These tests included: End-to-end tests for generating and posting comments.

### End-to-End Testing for Comment Generation

**Objective**: Ensure that the extension can generate and post comments on Tumblr posts.

**Procedure**:

1. Load the Tumblr page with the extension enabled.
2. Click the "Reply" button on a post to trigger the comment box insertion.
3. Select a tone and click "Generate Comment".
4. Verify that the comment is generated and displayed in the text area.
5. Click "Post Comment" and ensure that the comment is posted to the Tumblr post.

**Toggle Functionality Testing (Beta)**

**Objective**: Verify that the toggle functionality enables and disables the extension.
**Procedure**:

1. Open the extension popup.
2. Toggle the switch to the "OFF" position.
3. Verify that the extension stops injecting the comment box on Tumblr posts.
4. Toggle the switch to the "ON" position.
5. Verify that the extension resumes injecting the comment box on Tumblr posts.

## 6.3 User Testing

User testing was conducted to gather feedback on the user interface and functionality. Several users tested the extension in different scenarios and provided feedback on their experience. Based on this feedback, improvements were made to enhance the user experience.

**User Feedback and Adjustments**

- **Feedback**: Some users found the comment box positioning to be inconsistent.
    - **Adjustment**: Improved the logic for positioning the comment box to ensure it appears consistently relative to the "Reply" button.
- **Feedback**: Users wanted more tone options for comment generation.
    - **Adjustment**: Added additional tone options to the dropdown menu in the comment box

## 6.4 Kanban Board Validation

The Kanban board was used to track all requirements and tasks for the project. Each task was marked as complete once it was implemented and tested. This served as a form of validation, ensuring that all project requirements were met.

**Key Requirements Met:**

- **Chrome Extension**: Successfully built and loaded as a Chrome extension.
- **Comment Generation**: Enabled users to generate AI-powered comments with selectable tones.
- **Toggle Functionality**: Implemented toggle functionality to enable and disable the extension.
- **API Integration**: Connected the frontend to the backend API to fetch generated comments.
- **User Interface**: Designed a user-friendly interface for the popup and comment box.

Here is a link to the kanban board to have a more detailed idea

# 7. Deployment

## 7.1 Building the Extension

The React app was built using Webpack, and the resulting build was used as the Chrome extension. The extension was tested locally before packaging for distribution.

## 7.2 Deploying the API

The Flask API was deployed on a local server for testing. Future plans include deploying the API to a cloud provider for better scalability and availability. However, using openai's or haiku APIs to get tailored responses would be better. A GPT agent that is free to use and can be customized for more thorough tests and API design is Free-Auto-GPT

# 7. Future Enhancements

Based on the feedback and observations during the testing phase, several key enhancements have been identified to further improve the functionality and user experience of TumblAI. These enhancements are crucial for optimizing performance, enhancing usability, and ensuring robustness across various scenarios.

## 7.1 Performance Improvements

### Reduce Latency for Comment Box Appearance

- **Objective**: Minimize the delay between when the reply section is opened and when the comment box appears.
- **Implementation**: Introduce a more efficient detection logic to identify when the reply section is opened. This will involve using MutationObservers more effectively to detect changes in the DOM and trigger the insertion of the comment box promptly.

### Fix Issues with Routing Between Pages

- **Objective**: Ensure seamless operation of the extension across different routes within the Tumblr dashboard.
- **Implementation**: Enhance the background script to monitor URL changes and send a `URL_CHANGED` message to the content script. This will reinitialize the observer and ensure the comment box functionality remains consistent across page transitions.

## 7.2 User Interface Enhancements

### Remove Previously Added Comment Box

- **Objective**: Prevent clutter and ensure a clean interface by removing previously added comment boxes when a new reply section is opened.

- **Implementation**: Implement logic in the content script to detect and remove any existing comment boxes before adding a new one. This will involve keeping track of currently active comment boxes and ensuring they are removed or hidden as needed.

**Implement Multiple Comment Boxes for Multiple Reply Sections**

- **Objective**: Allow users to interact with multiple reply sections simultaneously by supporting multiple comment boxes.
- **Implementation**: Adjust the content script to handle multiple instances of comment boxes. This will involve creating a unique identifier for each comment box and ensuring they are managed independently.

## 7.3 AI Model Integration

**Tailored AI Responses**

- **Objective**: Enhance the relevance and context of generated comments by integrating more sophisticated AI models.
- **Implementation**: Replace the predefined responses with AI-generated content that is tailored to the specific context of the post and the selected tone. This will involve integrating an AI service such as OpenAI's GPT-3 or similar models.

## 7.4 Enhanced User Customization

**Customizable Tones and Responses**

- **Objective**: Allow users to define and customize their own tones and responses.
- **Implementation**: Provide a user interface within the extension settings where users can add, edit, and delete custom tones and their corresponding responses. Store these customizations locally and ensure they are applied during comment generation.

## 7.5 Robust Error Handling and Logging

**Comprehensive Error Logging**

- **Objective**: Improve error detection and troubleshooting by implementing comprehensive error logging.
- **Implementation**: Integrate a logging framework to capture and report errors. This will include both frontend and backend errors, providing detailed information to aid in debugging and improving the extension.

# Conclusion

The TumblAI project successfully demonstrates the potential of AI in enhancing user engagement on social media platforms. The Chrome extension provides a seamless experience for generating and posting comments on Tumblr, making interactions more dynamic and engaging. Future enhancements and continued development will further improve the functionality and user experience of TumblAI.

The planned enhancements for TumblAI are aimed at improving performance, enhancing the user interface, and providing more relevant AI-generated content. By addressing the identified issues and implementing these enhancements, TumblAI will offer a more robust, efficient, and user-friendly experience. The focus on user customization and AI integration will further enhance the extension's value, making it a powerful tool for engaging with Tumblr posts.

Check out the repo here for more details about installation and to take a deep look at the code.