TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

You probably have to follow the structure of this first paragraph.

The thesis titled **Fission in the Frontend: Building a Nuclear Reactor in JavaScript** was developed by student LastName FirstName as a bachelor's project at the Technical University of Moldova. It is written in English and contains an introduction, 4 chapters (Domain Analysis, Requirements Specification, System Design, System Implementation), a list of figures, a list of tables, conclusion, bibliography, and appendices.

Keywords: JavaScript, nuclear reactor, frontend engineering, asynchronous meltdown, satire

In this groundbreaking study, we explore the totally safe and not-at-all terrifying proposition of building and operating a fully functional nuclear reactor system entirely in JavaScript. Historically relegated to the humble browser and the occasional web app that crashes if you press Backspace too hard, JavaScript is the ideal language for controlling a high-risk, high-complexity critical infrastructure system like a nuclear power plant. Leveraging cutting-edge technologies like setTimeout(), localStorage, and a loosely defined global state, we demonstrate how a single-page web app can reach critical mass—both figuratively and literally. With asynchronous core cooling (using Promises), CSS-based hazard lights, and a Node.js-powered Geiger counter API (because why not), our system boasts at least 60% uptime and 40% existential dread. This paper is a satirical ode to the modern web stack and its uncanny ability to appear everywhere it absolutely should not.

INTRODUCTION

Disclaimer: This template is not an official reference; there might be mistakes. If you find any inconsistencies, go on and share updates to this template (e.g. pull requests). Only through combined efforts will we be able to achieve the perfect template that the advisers will finally be happy with.

1 DOMAIN ANALYSIS

There must be at least a single paragraph between a chapter and a section. Here you can simply summarize the chapter, like ”In this chapter we will do this and that”.

1.1 Problem Definition

Here you explain why you’ve decided to do this thesis project. Was there no existing solution for a problem? Did you have an idea on how to improve an existing solution? Or perhaps you simply wanted to check a hypothesis, like ”is JS a good language for building a nuclear reactor”?

Notice how the section titles use Title Case. We weren’t given any strict rules, but basically:

- capitalize the principal words;
- capitalize prepositions and conjunctions of four letters or more;
- lowercase the articles the, a, and an.

By the way, here is how to write simple lists:

- first later is lowercase;
- semicolon at the end;
- period at the end of last item.

1.2 Some Technical Concept

The next section can be an introduction to some technical concepts that your topic requires. Use diagrams, either your own or from the internet (in which case you have to cite them).

Floats (figures, tables and code listings) are not allowed to be at the top of the page, there should be at least a few lines separating them from the top. Also, they may not touch each other, there should be at least a few lines separating them. Look up how to use `[!hb]` after `\begin{figure}`, this way Latex will aid you in placing them according to these rules.

1.2.1 Subsection Example

A figure should appear in the same section it’s referenced in, so be careful that Latex doesn’t place it in the next section because it couldn’t fit in the current one. It can be placed before it is referenced though (I think, correct this if I’m wrong). This is one reason why sections should span at least one page, so that you have enough room to place the floating figures within it. If your sections are too small, simply merge them.

Notice how figure 1.1 couldn’t fit on the page right after the section title where it was defined, so it

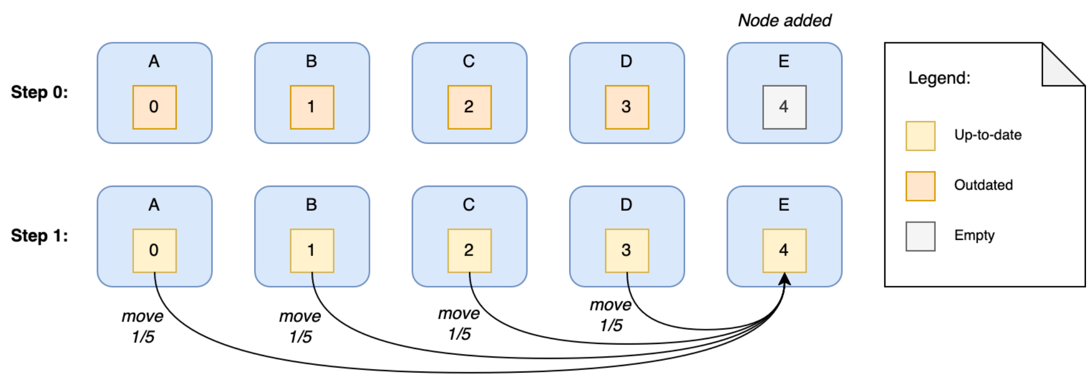

Figure 1.1 - The Diagram Captions Are Also Title Case [1]

was moved to the next page instead. But now it's at the top of the page, which UTM doesn't allow. So you have to move the definition of the figure somewhere else, in this case after this paragraph. Setting `[!hb]` on the figure would make it go to the bottom of the page, leaving a big gap between this text and the figure, so that's not a solution.

1.3 How to write Mathematics

LaTeX is great at typesetting mathematics. Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables with $\mathrm{E}[X_i] = \mu$ and $\mathrm{Var}[X_i] = \sigma^2 < \infty$, and let

$$S_n = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{i}^{n} X_i$$

denote their mean. Then as n approaches infinity, the random variables $\sqrt{n}(S_n - \mu)$ converge in distribution to a normal $\mathcal{N}(0, \sigma^2)$.

1.4 How to add Citations and a References List

You can simply upload a `.bib` file containing your BibTeX entries, created with a tool such as JabRef. You can then cite entries from it, like this: [1]. Just remember to specify a bibliography style, as well as the filename of the `.bib`. You can find a video tutorial here to learn more about BibTeX.

If you have an upgraded account, you can also import your Mendeley or Zotero library directly as a `.bib` file, via the upload menu in the file-tree.

2 REQUIREMENTS SPECIFICATION

In any mission-critical system — especially one that casually toys with the forces that power the sun — a clearly defined set of requirements is not just helpful; it's the thin red line between a safe, controlled simulation and an all-you-can-explode buffet. This chapter delineates the specifications that our JavaScript-powered nuclear reactor absolutely must meet to function (somewhat) reliably without turning the entire browser tab into a radioactive wasteland. While the boundaries between software engineering and nuclear physics remain politely blurred here, our goal is to capture the essence of reactor control within the gloriously quirky constraints of the modern web stack.

2.1 Functional Requirements

You can put your use-case diagrams in this section. But I don't know for sure, again, ask the coordinators. Once you know for sure, edit this paragraph and make a pull request. You can do it, I believe in you! "See my diagrams in Figure 2.1 and Figure 2.2."

Figure 2.1 - You Have To Manually Set the Width of Each Picture

Here is an example of an UTM-style enumeration. UTM only allows unnumbered list (itemizations), but we are allowed to label the items this way:

- **FR 1.** The system shall simulate nuclear fission reactions using JavaScript's event loop to model neutron interactions asynchronously.
- **FR 2.** The system shall provide real-time status updates of core temperature, radiation levels, and coolant flow via a dynamic web interface.
- **FR 3.** The system shall allow the user to initiate emergency shutdown procedures (SCRAM) through a dedicated button that triggers cascading Promise cancellations.
- **FR 4.** The system shall implement a virtual control rod insertion mechanism controlled by draggable UI components, adjusting reactivity on the fly.
- **FR 5.** The system shall log all critical events and state changes to the browser's localStorage for

post-meltdown forensic analysis.

- **FR 6.** The system shall visualize hazard warnings using CSS animations, flashing red alerts whenever safety thresholds are exceeded.
- **FR 7.** The system shall provide an API endpoint (Node.js) that streams simulated Geiger counter data to subscribed clients in real time.

Notice how I moved the Figure 2.2 farther down so it wouldn't stick to the previous figure, even though it was mentioned in the first paragraph of this section. Unfortunately, there is no way to tell latex "put this image somewhere in this section, but never at the top of the page or touching another figure. You can only tell it "put it here or somewhere at the bottom of the page"

Figure 2.2 - Don't Forget to Cite Images from the Internet Like This [1]

Also you better have text between captions and section titles.

2.2 Non-Functional Requirements

A list of non-functional requirements, you know.

3 SYSTEM DESIGN

Here is a Table 3.1.

Table 3.1 - Example Table That Fits Page Width

Column 1	**Column 2**	**Column 3**
Long wrapped content that fits nicely in the cell and breaks into multiple lines	Another wrapped text entry to demonstrate alignment	Final column with enough text to require wrapping too
Short entry	Medium text content here	More wrapped content

4 SYSTEM IMPLEMENTATION

Paragraph before code.

```
let reactorCoreTemp = 0;
const maxTemp = "9000";
const cooling = true;

function startReactor() {
  while (reactorCoreTemp < maxTemp) {
    reactorCoreTemp += Math.random() * 10;
    if (cooling = false) {
      reactorCoreTemp -= 50;
    }
    setTimeout(() => console.log("Core temp:", reactorCoreTemp), 1000);
  }
  alert("Critical temperature reached! Initiate meltdown protocol.");
}

startReactor();
```

Listing 4.1 - Main Reactor Loop

Paragraph after code (check the spacing).

Figure 4.1 - One more frog for good measure

CONCLUSIONS

Bibliography

1. GREENWADE, George D. “The Comprehensive Tex Archive Network (CTAN)”. In: *TUGBoat* 14.3 (1993), pp. 342–351. URL: `www.cornhub.com`.

APPENDIX A

Some code

```javascript
// Tokenize input string into Lisp tokens
const tokenize = input =>
  input
    .replace(/\(/g, " ( ")
    .replace(/\)/g, " ) ")
    .trim()
    .split(/\s+/);

// Parse tokens into AST
const parseTokens = tokens => {
  if (tokens.length === 0) {
    throw new SyntaxError("Unexpected EOF while reading");
  }
  const token = tokens.shift();
  if (token === "(") {
    const list = [];
    while (tokens[0] !== ")") {
      list.push(parseTokens(tokens));
      if (tokens.length === 0) {
        throw new SyntaxError("Unexpected EOF while reading");
      }
    }
    tokens.shift(); // Remove ')'
    return list;
  } else if (token === ")") {
    throw new SyntaxError("Unexpected )");
  } else {
    return atom(token);
  }
};

// Convert token to number or symbol
const atom = token => {
  const num = Number(token);
  return isNaN(num) ? token : num;
};

// Top-level parse function
const parseLisp = input => parseTokens(tokenize(input));

// Example usage:
const code = "(define (square x) (* x x))";
const ast = parseLisp(code);
console.log(JSON.stringify(ast, null, 2));
```