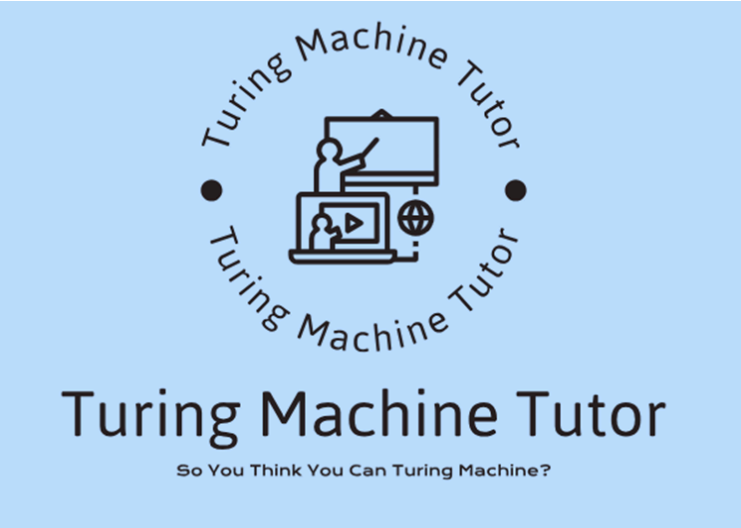


Technical Advisor prof. Dana Fisman

Academic Advisor prof. Gera Weiss

Team Members

Bayan Kassem

Radwan Ganim

Yuval Dolev

Meitar Daddon

Table Of Contents

Chapter 1- introduction

1.1 Problem domain

Some of the computer science concepts and computational models are hard to understand , explain and visualize. Our project will overcome these struggles using interactive web application and animations.

1.2 context

System Context:

1. User Interface: The web app provides an interactive user interface where users can design and test turing machines. It includes areas for building Turing Machines and running tests.

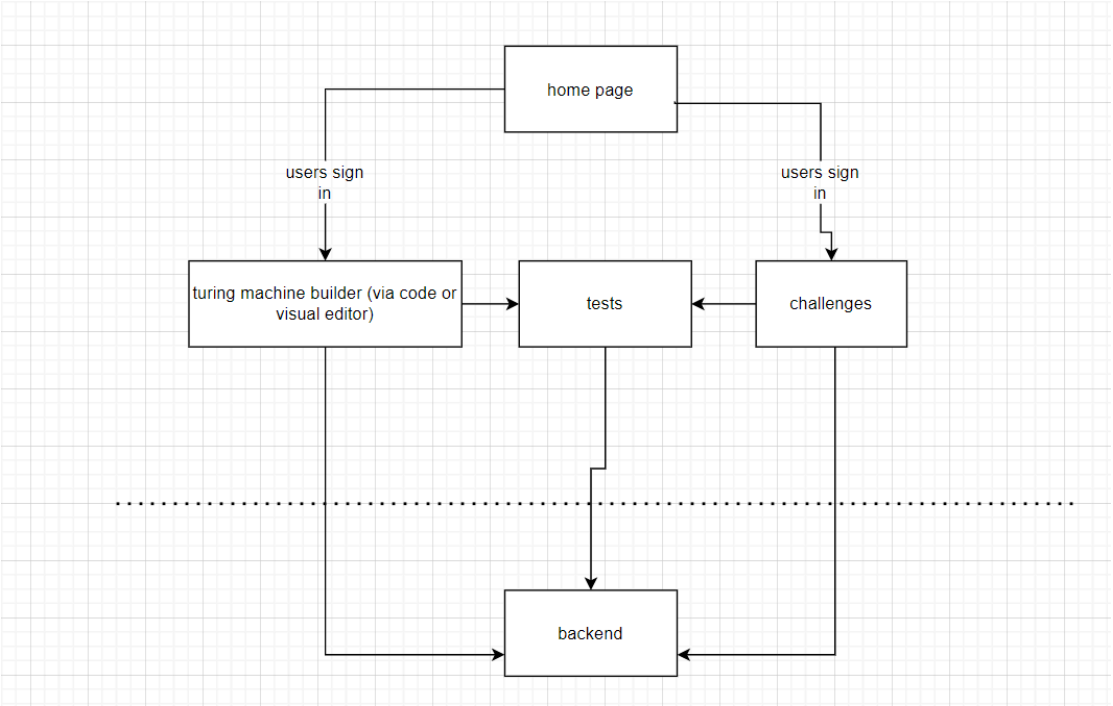
2. turing machine Builder: Users can create and modify different types of formal language decision models, using a visual/programming-based editor. Other concepts from the “computational models” course may be added in the future, depending on time constraints; however, the primary focus is on Turing Machines.

3.challenges: it is nice to have a challenge tab, where lecturers add challenging languages and users try to make a turing machine for the languages.

4. Test Runner: The system allows users to input test cases, and it runs simulations of the turing machines, providing step-by-step visualizations of the turing machine's behavior.

5. Backend Server: Handles the logic for parsing, simulating, and animating the turing machine. It communicates with the front-end and may store user-created turing machine for later use.

Block Diagram:



Components:

1. User Interface: Provides a graphical interface for users to design turing machines, input test cases, and view results.

2. turing machine Builder: python and/or JS/TS code editor, ~~and/or visual editor~~ for building Turing Machines. Allows users to define states, transitions, and other relevant details ~~visually / or~~ by code.

3. Test Runner: Accepts user-inputted test cases, runs simulations on the chosen turing machine, and displays animated models showing the turing machine behavior step by step.

4. challenges: allows mainly lecturers to add challenging languages that users can compete with each other to make the most suitable turing machine.

5. Backend Server: Manages the logic for parsing user inputs, simulating turing machines, and generating animations. It may also handle user accounts and store user-created turing machines for future use.

Interconnections:

- The User Interface communicates with the turing machine Builder to create and modify turing machines visually.
- Test Runner takes input from the User Interface and communicates with the Backend Server for simulation and animation logic.
- The Backend Server manages the overall logic, including parsing, simulating, and animating the turing machine.

External Interfaces:

- Users interact with the system through the User Interface, where they build turing machines, input test cases, and view results.
- The Backend Server interfaces with the front-end and handles the logic for simulation, animation, and potentially user account management.

1.3 vision

The app will visualize concepts such as Turing machines.

The app will allow students to construct concrete examples of those models using code and then run them on inputs and watch the process.

The app will present students with challenges in the forms of formal languages that they then should create code that creates models that identify these languages.

The app will help the staff of the course computational models to better demonstrate and explain material and the sophisticated models.

1.4 stakeholders

Our targeted audience in general are people who are interested in deep and tangible understating in some sophisticated models like Turing machines, and in particular students and staff of the course computational models, so prof. Dana Fisman is our client and our provider of the requirements.

1.5 software context

Our project will be implemented as web application, it will have a backend and frontend , it will be able to process python code that will be provided by the user, check the correctness of models and display the results, also it will be able to visualize and show animations of turing machines that will be provided by the user.

major inputs:

1. User Input: Users input specifications for Turing Machines using one of the options in the subsections of this requirements. Users provide test cases to evaluate the behavior of the created turing machine. Ways to specify the Turing Machine will be chosen in a later specification step, after risk-assessment for each, from the following options:
    a. Code editor using high-level programming language (such as Python) to specify an algorithm for constructing the Turing machine formally
    b. A visual editor for a **costum, domain-specific visual-language** (AKA “block programming” or “no-code”; that is, a visual language in the style of Scratch or LabView) that allows to specify an algorithm for constructing the Turing machine.

Functionality:

1. Turing Machines Building:

- Users can visually design Turing Machines through python code / visual editors.
- Functionality includes adding states, defining transitions, and setting accepting or final states. (by code)

2. Test Execution:
    - Users input test cases to evaluate the behavior of their created turing machine.
    - The system runs simulations on the chosen turing machine to process the input and generate results.
    - [Optional, NTH] for formal languages (“challenges”), the educators authoring the examples may provide high-level language implementation of checking whether a word is in the language. This may allow the app to generate random tests and compare them to the user’s model results, thus guaranteeing further confidence that the user's model indeed solves the language and not just coincidentally fits for a set of known unit-tests.
        - Alternatively [or additionally], maybe allow the students to write such code too, in order to prime them towards solving the same language in Turing machines

3. Animated Models[NTH]:
    - The system provides animated models illustrating the step-by-step execution of the turing machine on input test cases.
    - Animation aids users in understanding the behavior of their turing machine.
4. Discussions:
    a. Users who have successfully solved a challenge get access to a per-challenge forum in which they can share their solutions and discuss complexity, simplicity and other ideas.

Processing:

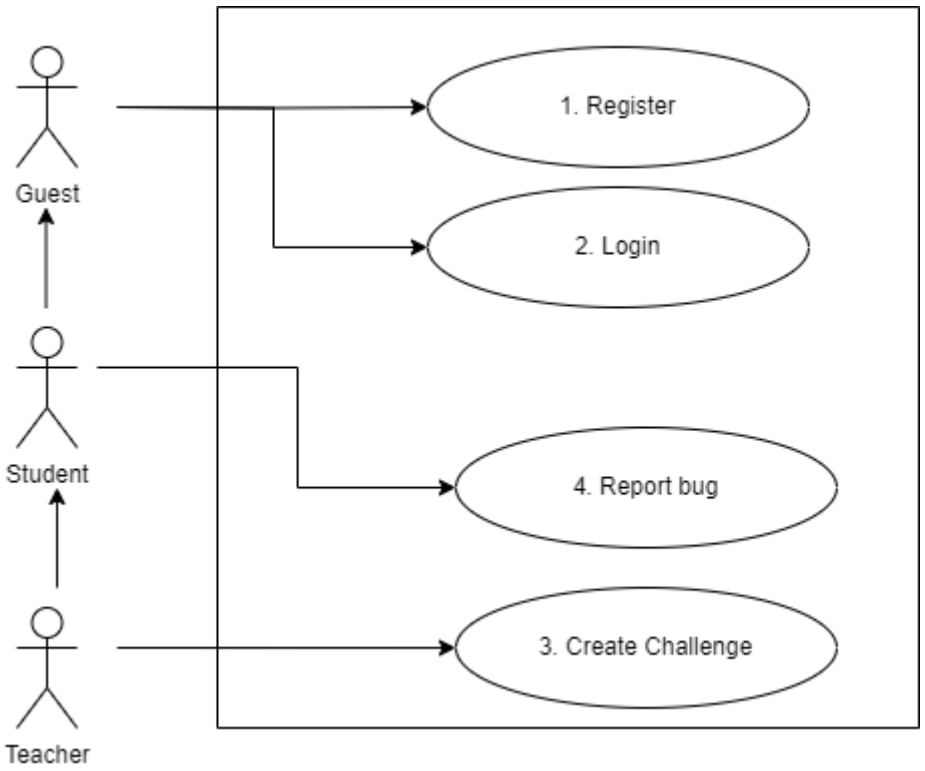
1. Turing Machine Parsing:
    - The system parses user-inputted specifications to create internal representations of Turing Machines.
2. Simulation Logic:
    - The system simulates turing machine behavior on provided test cases, tracking state transitions and processing steps.
3. Animation Generation:
    - Animated models are generated to visually represent the execution of a Turing Machine on input test cases.

Outputs:

1. Test Results:
    - The system outputs results of test cases, indicating whether the turing machine accepts or rejects each input.
2. Animated Models:

- Visual representation of a turing machine execution on input test cases, aiding users in understanding their Turing Machine‘s behavior.

Use Cases: Common Usage Scenarios:



Use case 1: User Registration

- **Actor**: User
- **Precondition**: User not logged in and not registered to the system
- **Postcondition**: System registered the user to the system
- **Parameter**: Credentials / Identifying details
- **Actions**:
    1. navigate to the registration page
    2. provide required information (username, email, password)
    3. submit the registration form
    4. system validates email is unique in the system
    5. system creates new member

Participants	Parameters	Expected Result	Scenario
User	Unique username, strong password	Successfully registered to the system	Good
User	username that already registered in the system, strong password	Username already used in the system, message will be thrown	Bad
user	invalid username or password - empty string , null , illegal name..	system throws an error describing the problem	sad

Use case 2: User Logs in

- **Actor**: User
- **Precondition**: User registered to the system before
- **Postcondition**: user logged in to the system
- **Parameter**: Credentials / Identifying details
- **Actions**:
    1. Enter username and password on the login page
    2. Click login button
    3. system validate the user's input
    4. user access the main dashboard upon successful login

Participants	Parameters	Expected Result	Scenario
User	username, password	Successfully logged in to the system	Good
User	username not in the system, password	Username not in the system, message will be thrown	Bad

case case 3: user creates a challenge

- **Actor**: User
- **Precondition**: User is registered to the system and he is a lecturer or admin
- **Postcondition**: challenge is created and posted
- **Parameter**: challenge details (name , expiry date,tests list,grades..)
- **Actions**:
    1. lecturer or admin logs in
    2. goes to challenges tab and clicks create new challenge
    3. fills in new challenge details
    4. clicks publish to launch the challenge

Participants	Parameters	Expected Result	Scenario
lecturer or admin	valid details(name , expiry date,tests list,grades)	Successfully publishing a challenge	Good
lecturer	malicious code	the system detects the attack , and prevents it using several	Bad

		techniques like honey pot..	
lecturer or admin	some invalid data in details - empty strings , due date in the past..	the system throws error message describing the error	sad

case case 4: user reports a bug

- **Actor**: User
- **Precondition**: User is registered to the system
- **Postcondition**: a bug report is submitted to the system
- **Parameter**:bug details - name of the bug , time of the occurrence of the bug, text describing how the bug occurred
- **Actions**:
    1. user logs in
    2. use clicks report a bug option
    3. user fills in bug details
    4. user submits the report
    5. email is sent to admins (the bug details)

Participants	Parameters	Expected Result	Scenario
user	valid details	Successfully reporting a bug	Good

user	malicious script code	the system detects the attack , and prevents it using several techniques	Bad
user	some invalid data in details - empty strings	the system throws error message describing the error	sad

Chapter 2 - usage scenarios

2.1 User Profiles - the actors

1. Student:

   Characteristics:

   - Typically enrolled in computer science, software engineering or related programs
   - has a basic understanding of Turing machines concepts
   - Uses the system for educational purposes, such as learning about Turing machines and their behaviors

   Interacting with the system:

- Creates Turing machines for training ,class assignments
- runs a simulations to understand the execution of different Turing machines
- Utilizes the system to visualize and experiment with turing machine concepts

2. Instructor:

   Characteristics:

   - Holds expertise in Turing machines theory and related subjects
   - Guides and teaches students in a formal educational setting
   - May use the system as teaching aid to demonstrate concepts

   Interacting with the system:

   - Creates sample Turing machines for instructional purposes
   - Demonstrate the use of the system during lectures or practical sessions

3. System Administrator:

   Characteristics:

   - Possesses technical expertise in system administration and maintenance
   - Responsible for ensuring the overall health and performance of the web app

   Interacting with the system:

   - Monitors system performance and user activity
   - Manage user accounts, permissions, and system configuration
   - Addresses technical issues, ensures data security, and performs routine maintenance

4. Researcher:

   Characteristics:

   - Engages in research related to turing machine theory, computation models
   - Seeks to experiment with and analyze various Turing machines configurations

   Interacting with the system:

   - Utilizes the system to model and test different types of Turing machines
   - Conducts experiments to gather data on the behavior of specific Turing machine
   - Extracts insights from the system for research publications

2.2 Use-cases:

Use case 1: Creating a new turing machine

- **Actor**: Member
- **Precondition**: Member in the system
- **Postcondition**: System creates the turing machine
- **Parameter**: python code
- **Actions**:
    1. navigate to the turing machine creation section
    2. Use the visual code editor to create the turing machine states, transitions and initial/final states
    3. Create the turing machine with a unique identifier

Participants	Parameters	Expected Result	Scenario
Member	username/id, code	Successfully created	Good
Member	empty string ,invalid code that does not compile	the system throws error message describing the error	Sad
Member	username/id, bad code	code with compilation error, message will be thrown	Bad

<u>Use case 2:</u> Editing an Existing Turing machine

- **Actor**: Member
- **Precondition**: Created turing machine by the member
- **Postcondition**: Turing machine updated
- **Parameter**: code
- **Actions**:
    1. Access the list of saved Turing machines
    2. selected desired turing machine
    3. user the visual code editor to make modifications
    4. Save the changes

Participants	Parameters	Expected Result	Scenario
Member	valid code	update turing machine code successfully and saved data	Good
Member	malicious code	the system detects the attack , and prevents it using several techniques like honey pot..	Bad
Member	invalid code that does not compile	error message that describes the error and the lines where the compile error occurred and turing machine is not saved	sad

case case 3: Saving a Turing machine

- **Actor**: member
- **Precondition**: Turing machine created
- **Postcondition**: system save the turing machine
- **Parameter**: user_id, code , id / name of the turing machine
- **Actions**:

1. Complete the creation or modification of a turing machine
2. choose to save the turing machine
3. provide a name or identifier for the saved turing machine
4. confirm save action

Participants	Parameters	Expected Result	Scenario
Member	code of turing machine and distinct id	turing machine code is saved	good
Member	code of turing machine and already used id or name	system throws a message describing the problem and how to solve	sad
Member	malicious code	system detect malicious code inputs and member is blocked from further usage of the system relevant message will be thrown	Bad

Use case 4: Running a Test on a turing machine

- **Actor**: member
- **Precondition**: Turing machine created

- **Postcondition**: system simulate the test's input on the turing machine
- **Parameter**: test input(string)
- **Actions**:
    1. Open the test execution interface
    2. input a test case for the selected turing machine
    3. initiate the simulation
    4. view the step-by-step execution and final result

Participants	Parameters	Expected Result	Scenario
Member	test input	the system runs the test and displays results	good
Member	test input , but the test makes infinite loop	the system runs the test, but after defined time it will throw error timeout and suggests common mistakes that may has occurred	sad
Member	test input not in the turing machine (abc / letters)	error message will be thrown	bad

Use case 5: Running multiple Tests

- **Actor**: member
- **Precondition**: Turing machine created
- **Postcondition**: system simulate the tests on the turing machine
- **Parameter**: tests list
- **Actions**:
    1. Select a turing machine for testing
    2. input a set of test cases
    3. initiate the tests
    4. review the results for each test case

Participants	Parameters	Expected Result	Scenario
Member	tests as lists	the system runs the tests and displays results	good
Member	tests as list , but at least one test makes infinite loop	the system runs the tests, but after defined time it will throw error timeout and suggests common mistakes that may has occurred	sad
Member	invalid tests input(contains letters not recognized by the turing machine)	error message will be thrown	bad

Use case 8: Retrieving a saved turing machine

- **Actor**: member
- **Precondition**: Turing machine created and saved
- **Postcondition**: system retrieves turing machine
- **Parameter**: turing machine id/name
- **Actions**:
    1. member enters turing machine saved list tab
    2. select previously saved turing machine
    3. system provide the user the relevant turing machine

Participants	Parameters	Expected Result	Scenario
Member	valid saved turing machine id/name	the system provide the user the turing machine code	good
Member	invalid saved turing machine id/name	error message will be thrown	sad
Member	valid saved turing machine id/name	database connection failed, error message will be thrown	bad

Use case 9: Viewing animated model

- **Actor**: member
- **Precondition**: valid turing machine
- **Postcondition**: animation and visualization of the turing machine
- **Parameter**: turing machine id ,word , tape , operator , r/ w head
- **Actions**:
    1. member access desired turing machine
    2. member provides which word to run on the machine
    3. the system will display the animation of the machine running the word

Participants	Parameters	Expected Result	Scenario
Member	word and id of turing machine	the system displays the animation of the machine while running the word	good
Member	invalid turing machine id	error message will be thrown	bad
Member	invalid word input(contains letters not recognized by the turing machine)	error message will be thrown	sad

Use case 10: Deleting a turing machine

- **Actor**: member
- **Precondition**: Turing machine created and saved
- **Postcondition**: system delete the saved turing machine
- **Parameter**: user_id, id / name of the turing machine
- **Actions**:
    1. user creates and saves new turing machine or accessing existing turing machine
    2. user selects to delete the turing machine
    3. system deletes the turing machine

Participants	Parameters	Expected Result	Scenario
Member	valid turing machine id/name	turing machine code is saved	good
Member	invalid turing machine id/name	relevant error message will be thrown	Bad

Use case 11: User Logout

- **Actor**: User
- **Precondition**: User registered to the system before and is currently logged in
- **Postcondition**: user logged out of the system
- **Parameter**: event logout button clicked
- **Actions**:
    1. user is currently logged in
    2. user selects to log out
    3. system successfully logs out the user

Participants	Parameters	Expected Result	Scenario
User	logout click event and user is currently logged in	Successfully logged in to the system	Good
User	logout click event and user currently not logged in	relevant error message will be thrown	sad
User	logout click event and user is currently logged in	system failed to update database, relevant message will be thrown	Bad

Use case 12: User edits Account settings

- **Actor**: user
- **Precondition**: valid user
- **Postcondition**: edited account settings
- **Parameter**: user email address, password, name
- **Actions**:
    1. user navigate to the account sittings
    2. user updated the desired information (email, password etc.)
    3. system saves the changes

Participants	Parameters	Expected Result	Scenario
Member	valid email address / password / name etc..	the system saves the new information and updates the database	good
Member	invalid email address ,empty strings , illegal passwords that are easy to guess	error message will be thrown	sad
Member	malicious code	the system detects the attack , and prevents it using several techniques like honey pot..	Bad

Use case 13: Language Selection

- **Actor**: User
- **Precondition**: User enters the home page (index.html)
- **Postcondition**: website language changed
- **Parameter**: desired language
- **Actions**:
    4. user select language tab
    5. user selects new language (english / hebrew)
    6. system successfully change the website language

Participants	Parameters	Expected Result	Scenario
User	english / hebrew	Successfully changed language	Good
User	not valid language (not in the list we implemented)	system failed to change language, language do not change, relevant error message will be thrown	Bad

Use case 14: Reviewing Test Results History

- **Actor**: member
- **Precondition**: Turing machine tests simulated before
- **Postcondition**: user review previous tests results
- **Parameter**: turing machine id/name
- **Actions**:
    1. select a turing machine
    2. access the result history section
    3. review the detailed results of past test cases

Participants	Parameters	Expected Result	Scenario
Member	valid turing machine id/name	the system shows all previous test cases results	good
Member	valid turing machine id/name but no tests has been simulated before or implemented	the system shows all previous test cases results	sad
Member	invalid turing machine id/name	error message will be thrown	bad

use case 15: System Administrator Monitoring

- **Actor**: User
- **Precondition**: user is administrator
- **Postcondition**: admin monitors system performance, and user activity and overall health of the system
- **Parameter**: desired dates activity
- **Actions**:
    1. user logs in and identifies as administrator
    2. admin access special tab, that is only visible to the admin
    3. selecting the desired date to view

Participants	Parameters	Expected Result	Scenario
admin	valid dates activity	the system displays the report	good
admin	invalid dates activity	error message will be thrown	sad
admin	malicious code	the system detects the attack , and blocks it.	Bad

2.3 Special usage considerations:

1. Multilingual Support:

   Requirement: The system should support multiple languages to cater to users from diverse linguistic backgrounds. (Hebrew, English)

   Rationale:

   Multilingual support enhances the usability of the system for users worldwide, making it accessible to a broader audience

2. Responsive Design:

   Requirement:

   The user interface must be designed to be responsive across different devices, including desktops, tablets, and mobile phones.

   Rationale:

   A responsive design ensures a consistent and optimal user experience, irrespective of the device used to access the system.

3. Security Measures:

   Requirement:

User data, especially Turing Machine specifications and results, must be stored securely. Encryption should be employed for data transmission between the client and server.

Rationale:

Security measures are essential to protect sensitive user data and ensure the confidentiality and integrity of information exchanged within the system.

4. Performance Optimization:

   Requirement:

   The system should be optimized for performance, ensuring that Turing Machine simulations and animated models run efficiently.

   Rationale: Performance optimization is necessary to provide users with a responsive and smooth experience, especially during complex simulations.

5. Data Recovery Mechanism:

   Requirement:

   Implement a robust data recovery mechanism to recover user data in case of unexpected system failures.

   Rationale:

   Data recovery mechanisms contribute to system reliability and prevent significant data loss in unforeseen circumstances.

6. Regular Backups:

   Requirement:

   Implement a regular backup mechanism to periodically back up user data, including Turing Machines and test results.

   Rationale:

   Regular backups are essential for data recovery in the event of system failures, ensuring minimal data loss and maintaining system reliability.

Chapter 3 -Functional Requirements

1. Turing Machine Builder:

   1.1. Create Turing Machine:

   - Users can create a new Turing Machine
   - Users define states, transitions, tape alphabet, and initial/final states using python code.

   1.2 Edit Turing machine:

   - Users can modify an existing Turing Machine
   - Editing capabilities include adding, modifying, or deleting states and transitions

2. Test Execution:

   2.1. Input Test Cases:

   - Users can input test cases for the turing machine
   - Test cases include input strings to evaluate the Turing machine's behavior

   2.2. Run Simulation:

- The system runs simulations on the turing machine for each test case.
- Simulations track state transitions and tape modifications.

3. Animated Models:

   3.1. Generate Animated Models:

   - The system generates animated models illustrating the step-by-step execution of the turing machine on input test cases
   - Animations show transition between states and modifications to the tape

4. Test Results:

   4.1. Output Detailed Results:

   - The system outputs detailed results for each test case.
   - Results include whether the turing machine accepts or rejects the input, state transitions, and tape modifications.

5. Turing Machine Saving and Retrieval:

   5.1. Save Turing Machine:

   - Users can save their created Turing Machines for future use.
   - Saved Turing machines include all defined states, transitions, and settings.

   5.2. Retrieve Turing Machine:

   - Users can retrieve and load previously saved Turing Machines.
   - The system should provide a user-friendly interface to manage saved Turing Machines.

6. User Authentication and Authorization:

   6.1. User Registration:

   - User can register for an account with the web app

   6.2. User Login:

   - Registered users can log in to the web app

   6.3. User Permissions:

   - Different user roles have appropriate permissions
   - Authorized users can edit and delete their saved Turing Machines
   - Authorized users can add and edit challenges.

7. User Interface and Interaction:

   7.1. Intuitive Design:

   - The user interface should be intuitive and user friendly

   7.2. Tooltips/Help Section:

   - Include tooltips or a help section to assist users in understanding the functionality

8. Documentations:

8.1. User Tutorials:

- Provide comprehensive tutorials on creating Turing Machines, running tests, and interpreting results.

8.2. Technical Documentation:

- Technical documentation for system administrators and developers should be available

9. Error Handling:

9.1. Robust Error Handling:

- The system should handle incorrect inputs gracefully
- Clear error messages should be provided to users in case of issues

10. Security:

10.1. Secure Authentication:

- User authentication should be secure
- Data transmission between the client and server should be encrypted

11. Compatibility:

11.1. Cross-Browser Compatibility:

- The web app should be compatible with popular web browsers

11.2. Responsive Design:

- Responsiveness should be maintained across different devices, including desktops, tablets, and mobile phones.

Chapter 4 - Non-functional requirements

The software should be implemented as a web application , not regular software, so users can just access the web site without the need to install anything.

4.1 Implementation constraints -

•Performance:

- The web app should ensure that Turing Machine simulations complete within a reasonable time frame
- Simulations for a given test should take no longer than 5 seconds on average.

- execution time / response time should be less than 5 seconds.
- throughput should be above 1000 users in any given time.

•Reliability & Stability

- The system should be designed to withstand potential server or network failures
- Data recovery mechanism should be in place
- storage capacity should be unlimited.
- if the server had a network failure , our system should know how to reconnect with the server and load the updated database.
- if a user had a network failure , the system should save his progress and recover it when he reconnects.

•Safety & Security

- User data especially Turing machine specifications must be stored securely
- Confidentiality is a constraint, all user data transmission must be encrypted
- Access to saved Turing Machines should be restricted based on user roles
- if we want to add users then their passwords should be encrypted
- Our web app should be script safe , which means malicious users can't inject malicious scripts to sabotage our system.

•Portability

- The web app should be deployable from common web browsers like Chrome, Firefox and Safari
- The system should support text in different languages (English, Hebrew)

•Usability:

- The system should be designed for users with the basic understanding of turing machine theory
- A user friendly interface should minimize the need for extensive training
- The users must have basic knowledge in coding (python).
- the users must have basic knowledge and know concepts about computational models.

•Availability:

- The system should aim for 24/7 availability with scheduled maintenance windows communicated in advance
- Factors affecting availability include server uptime, network stability, and database reliability

4.2 Platform constraints

The system must be implemented as a web application as requested from our client , there for the system should be developed using technologies compatible with common web browsers (HTML, CSS, JavaScript), also The System should be deployable on cloud platforms like AWS or Azure.

Database Management should use a relational database system, such as MySQL, MSS or PostgreSQL.

we are considering some options:

1) backend implementation in python and front end in javascript, the challenge here is how to make the interaction between two different languages, we know that is it possible to make the interaction.
2) implementing using only python , we can use flask , which is a python library, for the frontend , and use sql alchemy as an ORM for the persistence layer.
3) or we can use only javascript .

4.2.1 SE project constraints

because of the fact that this is going to be a web application , so we gonna need a domain to run the server at (not just local host) , the system will be interactive , the majority of the inputs will come from the users and some will be injected by the developers (the tests and the available formal languages) , the system won't require any special hardware or access to secret data, our goal is to make a demo for the system so we can share it with the students and staff of the course computational models in the next semester , and in addition to our tests , we will add bug report system so that the users that use the beta version can report about bugs so we can fix it.

4.3 Special restrictions & limitations

The system assumes that users have a basic understanding of Turing Machines concepts.

The system should adhere to web development standards and guidelines

Chapter 5 -Risk assessment & Plan for the proof of concept

1. Technical Risks:

Risk: Incompatibility with Chosen Technologies

- solution: Conduct thorough research on technologies. Choose well-documented and widely supported frameworks.
- Plan: Allocate time for technology exploration and experimentation during the PoC phase.

Risk: Performance Bottlenecks

- solution: Develop a prototype with a focus on performance testing. Identify potential bottlenecks early.
- Plan: Implement key features to stress test the system's performance and scalability.

Risk: Integration Challenges

- solution: Choose technologies with good integration capabilities. Develop prototype components with modularity in mind.
- Plan: Include a simplified version of the integration points in the PoC.

2. Requirement Understanding Risks:

Risk: misunderstanding of Requirements

- solution: Regularly communicate with stakeholders and our client for clarification, after a defined phase of time meet with client to show progress for feedback.
- Plan: Begin the PoC with a well-defined subset of requirements and incrementally expand.

Risk: Evolving Requirements

- solution: Engage in continuous communication with stakeholders. Implement flexible design patterns to deal with changes.
- Plan: Plan for regular feedback sessions with stakeholders during the PoC.

3. Design Decision Risks:

Risk: Poor Design Decisions

- solution: Involve our academic advisor in the decision-making process.
- Plan: Allocate time for architectural discussions and prototype design.

Risk: bad User Experience

- solution: Involve project course staff and client in the design process. Prototype user interfaces to gather early feedback.

- Plan: Develop a basic user interface as part of the PoC to demonstrate the intended look-and-feel.

5. Project Failure Risks:

Risk: Insufficient Risk Reduction

- solution: Prioritize high-risk areas in the PoC. Perform basic testing and validation of critical functionalities.
- Plan: Regularly review risk solutions strategies and adjust as needed.

Risk: Lack of Stakeholder Engagement

- solution: Maintain open communication channels with stakeholders and clients. Organize regular meetings and demos.
- Plan: Include stakeholders and clients in PoC milestones and demonstrations to keep them up to date.

Conclusion:

The PoC plan is designed to address potential risks by emphasizing early validation, iterative development, and continuous stakeholder and client involvement. By actively seeking feedback, testing critical functionalities, and using the PoC to inform design decisions, the team aims to reduce the risk of project failure and build a solid foundation for the final system design.

Appendices:

technical information about the project:

• I/O format information:

- File Formats: Turing machine tutor will support importing and exporting Turing Machine configuration in a standardized JSON format to facilitate interoperability with other tools and systems
- Input Data Format:Test Cases will be provided through a user-friendly interface, allowing users to enter input data directly.

• Cost analysis studies:

- Cost implications: A preliminary cost analysis has been conducted, considering development resources, infrastructure, and maintenance costs. The aim is to optimize resource allocation and stay within budget constraints.

• user surveys:

- User Needs: Initial user surveys might be conducted to gather insights into user expectations, preferences, and pain points related to automata theory learning tools. This feedback may influence certain design choices.

• Glossary:

- Turing Machine (TM): A theoretical computing device consisting of an infinite tape and a read/write head that moves along the tape, capable of performing computations following a set of defined rules.
- Computational Theory: The study of abstract machines and their computational abilities.
- Proof-of-Concept (PoC): A prototype implementation designed to test and validate specific technical aspects of the system.
- JSON (JavaScript Object Notation): A lightweight data interchange format used for easy data exchange between systems.
- Interoperability: The ability of software components or systems to work together and exchange information seamlessly.