Oregon State University
College of Engineering
Electrical Engineering and Computer Science

Software Innovation Lab
Master's Project Report

Supreeth Suresh Avadhani

METADEV
Metadata Driven ERP Development Framework

Defended 14th June, 2023
Commencement June, 2023

Abstract

Metadev is an open-source framework (https://www.npmjs.com/package/mv-core) that facilitates the generation of complete Enterprise Resource Planning (ERP) applications through metadata. By ‘metadata‘ I mean module-specification files from which Metadev produces a full-stack ERP system. ERP systems are a type of software which helps businesses to manage their day-to-day operations; for example, an ERP might be used to maintain employee records. The metadata can be written manually, generated from a text prompt, or using Metadev’s Graphical User Interface (GUI). The framework’s goal is to make implementing ERP systems affordable and simple for small businesses and non-profit organizations.

Acknowledgments

I would like to begin by expressing my deepest gratitude to Professor Will Braynen, whose expertise, guidance, and unwavering support have been invaluable throughout this project. His mentorship and insightful feedback have significantly contributed to the quality and depth of my research.

Next, I want to acknowledge my parents, Dr. N. Suresh and Dr. Uma Suresh, for their constant love, encouragement, and sacrifices. Their belief in my abilities and unwavering support have been a driving force behind my achievements. I am eternally grateful for their faith in me.

To my dear brother and sister-in-law, Sumanth and Dr. Lekha, thank you for always being there, providing me with a listening ear, and offering words of encouragement. You prepared me for this master's journey technically and emotionally even before I started the program.

I am deeply grateful to my grandad, Dr. J. Venkataramana, and grandmom, Lakshmi Ramana, for their blessings, love, and wisdom. Their unwavering support and guidance have been a source of strength and inspiration throughout my academic journey.

I would like to express my heartfelt appreciation to my aunts, Dr. Anu Canumalla and Latha Subramanian, and uncles, Dr. Sridhar Canumalla and Subbu Srinivasan. Their support, encouragement, and belief in my abilities have provided me with the much-needed moral and emotional backing during challenging times. They are my home away from home.

Special thanks go to my mentor and main source of motivation, Raghu Bhandi. Your guidance, expertise, and constant encouragement have been instrumental in shaping the direction of my research. Your dedication and belief in my potential have propelled me to push my boundaries and strive for excellence.

Lastly, I want to express my heartfelt appreciation to my girlfriend, Tejashree. Your unwavering support, love, and understanding have been a constant source of strength and inspiration to me.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Overview

Metadev is an open-source framework that generates frontend and backend code for Enterprise Resource Planning (ERP) applications with a simple text prompt. ERP refers to a type of software that organizations use to manage day-to-day business activities such as accounting, procurement, project management, risk management and compliance, and supply chain operations. Metadev addresses the challenges of building ERP systems by striking the right balance between having a user-friendly interface and offering customization options.

In this chapter, I discuss the costs and benefits of ERP systems (section 1.2), the common features of such systems (section 1.3) and their prevalent architecture (section 1.4). This sets the groundwork for Chapter 2, which delves into how Metadev streamlines ERP application creation for both technical and non-technical users, ensuring tailored solutions fit specific project needs.

1.2 Costs and Benefits of an ERP System

ERP (Enterprise Resource Planning) systems are designed to manage and integrate key business processes, including accounting, procurement, human resources, and inventory. Implementing an ERP system can offer organizations numerous advantages such as streamlined processes, heightened productivity, enhanced decision-making, and improved visibility into operations.

To stay competitive, an increasing number of businesses have adopted ERP systems over the years. The global ERP software market was valued at $40 billion in 2020 and is projected to reach $47.5 billion by 2023 [1]. The rising demand for cloud-based ERP solutions, known for their flexibility and scalability, is fueling this growth.

The cost and duration of ERP system implementation can vary based on organizational scale and complexity, system choice, and required customization. On average, ERP implementation costs $3.7 million and takes 14.3 months [2]. However, these figures can be considerably higher for larger organizations or more complex implementations.

Despite the costs and time involved, ERP systems can yield substantial long-term benefits. They can bolster a business's bottom line by enhancing efficiency, cutting costs, and refining customer service. A NetSuite study revealed that 67% of businesses with an ERP system reported increased overall efficiency, 50% reported cost reductions, and 45% cited improved customer service [3].

The ERP system development process typically encompasses several phases: requirements gathering, system design, development, testing, deployment, and maintenance. These phases can be broken down further into project initiation, planning, execution, monitoring and controlling, and closure.

While ERP systems have demonstrated their capability to offer businesses a competitive edge and optimize operations, they demand significant time and resources for development and implementation. Although large enterprises might afford such systems, smaller businesses, entrepreneurs, and non-governmental organizations might find it challenging. Metadev revolutionizes this landscape by making ERP development and implementation both affordable and user-friendly for businesses of all sizes.

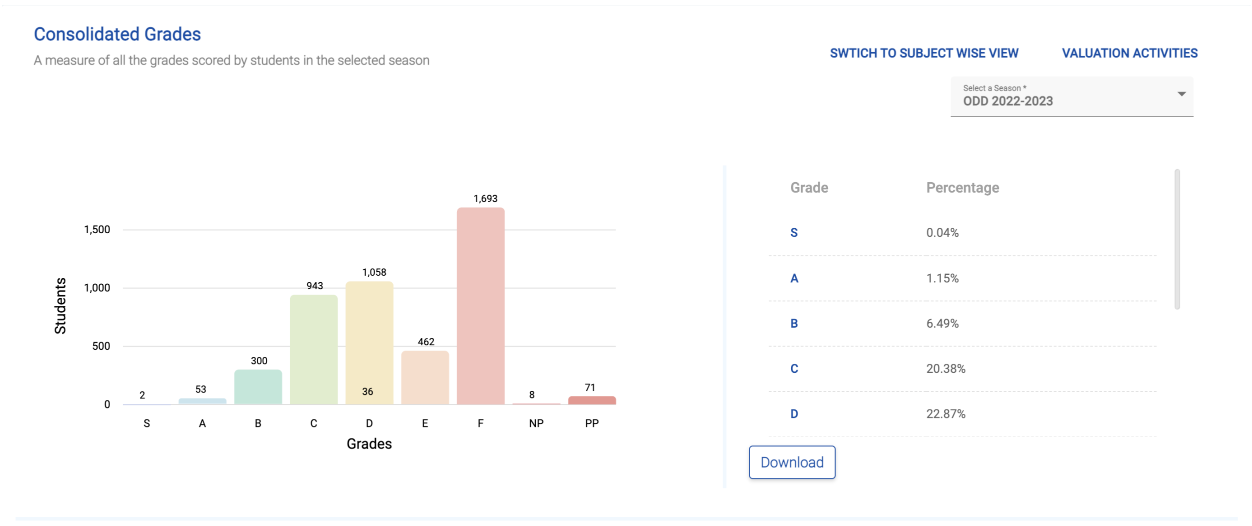
1.3 Common Features of ERP Systems

ERP systems typically comprise a vast array of screens. In practice, the number of screens can range from as few as several dozen to several thousand. Nonetheless, the majority of these screens follow a specific pattern. An authenticated user within an ERP usually performs the following operations:

1. View Data
2. Add Data
3. Modify Data
4. Delete Data

To facilitate these operations, ERP systems employ the subsequent components:

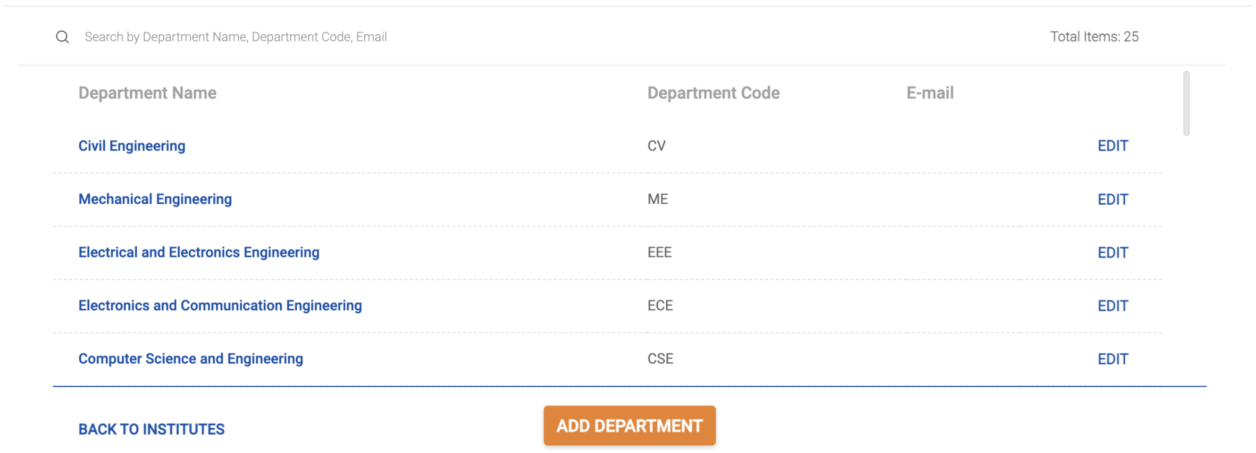
1. Tables
2. Charts
3. Forms
4. Pre-filled Forms
5. Navigation Menu



Grade	Percentage
S	0.04%
A	1.15%
B	6.49%
C	20.38%
D	22.87%

Figure 1.1: A screen to view data using a table and chart

These figures are sourced from an ERP currently in use by educational institutions. Figures 1.1 to 1.4 provide a visual representation of ERP operations in a real-world context, highlighting the components discussed in this section.

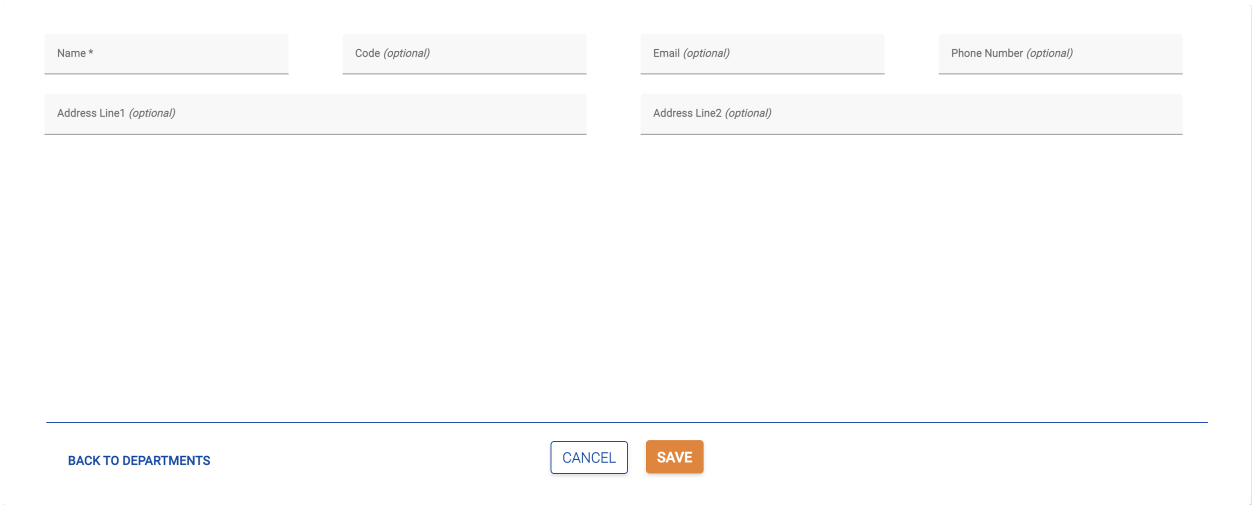


Search by Department Name, Department Code, Email

Total Items: 25

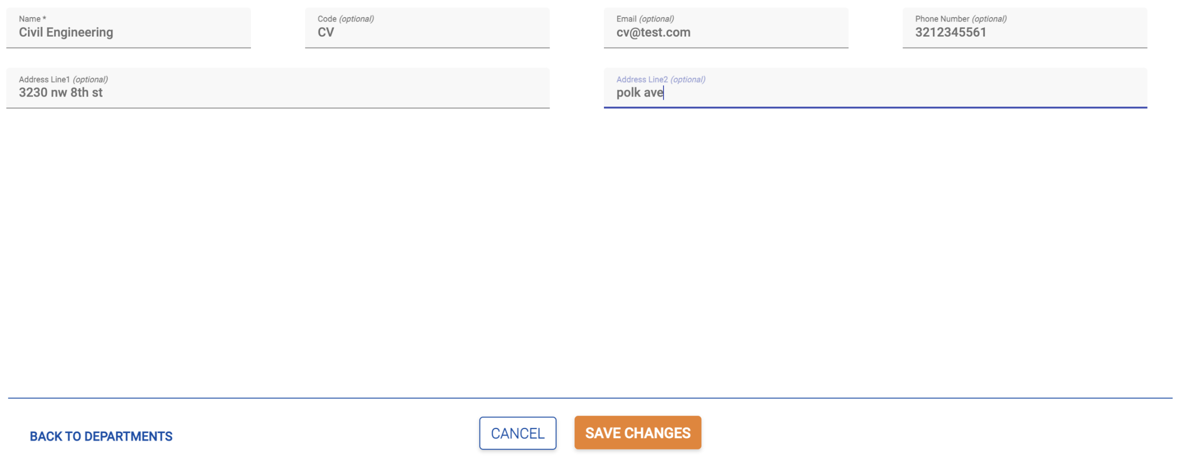
Department Name	Department Code	E-mail	
Civil Engineering	CV		EDIT
Mechanical Engineering	ME		EDIT
Electrical and Electronics Engineering	EEE		EDIT
Electronics and Communication Engineering	ECE		EDIT
Computer Science and Engineering	CSE		EDIT

Figure 1.2: A table view screen with options to search and filter



Name *	Code *(optional)*	Email *(optional)*	Phone Number *(optional)*
Address Line1 *(optional)*		Address Line2 *(optional)*	

Figure 1.3: A form screen for adding data



Name * Civil Engineering	Code *(optional)* CV	Email *(optional)* cv@test.com	Phone Number *(optional)* 3212345561
Address Line1 *(optional)* 3230 nw 8th st		Address Line2 *(optional)* polk ave	

Figure 1.4: A pre-filled form screen for editing data

Metadev enhances the development process of these patterns by offering a streamlined approach. It presents a graphical user interface for component generation and also produces the complementary server-side code for the created user interface.

1.4 Prevalent Architecture of ERP Systems

The N-tier architecture pattern, dividing applications into client and server sides, is the dominant blueprint for ERP applications due to its consistent performance in meeting user needs.

1.4.1 Client Side

Designed to amplify user satisfaction, the client side stresses a front-to-back method to guarantee a user-centric interface. It communicates with the server side through common protocols for efficient data transfer, emphasizing usability and performance.

1.4.2 Server Side

The server side acts as a central "System of Records" meeting statutory and managerial demands, ensuring data precision. Additionally, it manages a myriad of services and can interface with other server applications or clients via standard protocols, allowing for adaptable communication methods.

1.4.3 Development Approach

Though the client and server sides are individual entities, their concurrent development ensures synergy and minimizes redundancy. Adhering to best practices like the DRY principle, both components, despite possible variance in data model complexity, maintain mutual domains and validation rules. This encourages uniformity, reusability, and ensures harmony between the two.

In summary, the prevailing method for crafting applications hinges on a dual-component architecture: the user-centric client side and the data-focused server side. Its widespread adoption stems from its capacity to meet user demands efficiently, fostering both robustness and user-friendliness.

2. Motivation

Metadev's inception centers around addressing challenges in software, notably ERP system development. Traditional methods, characterized by manual coding, often consume time and are susceptible to errors. Metadev counters this with an intuitive graphical user interface and automated code generation. Research indicates that tools like Metadev can potentially cut development times by up to 40

A salient motivation for Metadev is development process simplification. Its visual interface permits developers to design and auto-generate code, boosting comprehension and efficiency – with potential time savings reaching 30% [4]. Metadev fosters software consistency and standardization through its pre-configured templates, enhancing code quality and maintainability [5].

Its accessibility stands out for non-tech experts, like business analysts, allowing them to visually craft and adjust systems, fostering better communication and reducing iterations [6]. For NGOs, Metadev is invaluable. With typically constrained resources, they benefit from its efficiency, directing more energy towards primary missions. Given the unique challenges NGOs encounter, Metadev's adaptability ensures software aligns with their specific demands.

Even for prominent tech firms, Metadev offers accelerated development, shortening product release times. Its generated code forms a baseline, upon which further customization can occur, ensuring software meets market needs with agility.

To conclude, Metadev's motivations span efficiency enhancement, development simplification, and adaptability, making it a fitting solution for both NGOs and large tech companies in their software endeavors.

3. The Problem Statement

This study delves into the drawbacks of conventional software development methods, especially in the realm of ERP systems. The current tools and approaches, encompassing manual coding, IDEs, and various platforms, are beset by issues related to time, intricacy, adaptability, and the inclusion of non-technical parties. These challenges compromise the development's efficacy, understandability, maintainability, and customization across different entities, such as NGOs and major tech corporations.

There's also the taxing concern of updating applications in line with swiftly advancing tech like React and Vue.js. Migrations not only bear heavy time and financial burdens but often don't align with projections. A study by Dimensional Research [7] underlines this, noting:

1. 57% of participants found migrations lengthier than anticipated.
2. 73% felt the transition spanned one to two years.
3. Budgets were exceeded in 55
4. 70% preferred multi-cloud strategies, driven by diverse benefits (77%) and avoiding sole vendor reliance (58

These findings contest the advantages touted by the dominant N-tier architecture pattern for ERP apps. This report introduces Metadev as a potential remedy, examining its capacity to elevate development practices. The goal is understanding if Metadev can foster development efficiency, uncomplicate procedures, endorse best standards, engage non-tech individuals, and allow tailoring. Through a comparative lens with established methods, this study aspires to gauge Metadev's merits in refining software creation and catering to an organization's specific needs.

4. Existing Solutions

Software development encompasses an array of solutions addressing the intricacies of constructing software, notably ERP systems. Each solution presents its advantages and inherent challenges. This section elucidates the prominent methods and their comparative merit.

Traditional manual coding, the bedrock of software creation, entails developers crafting code from the ground up. This avenue, while affording comprehensive control and customization, is susceptible to errors and extended development periods [8].

Integrated development environments (IDEs) amplify developer productivity with a fusion of code editors, debuggers, and project management utilities. However, they're intrinsically rooted in manual coding, devoid of visual interfaces akin to Metadev's offerings [4].

Emerging platforms advocating low-code and no-code paradigms empower users to fabricate software with scant to zero coding. Although rapid in development, they potentially compromise on adaptability and intricate customizations [5].

Established commercial ERP systems, epitomized by SAP, Oracle, and Microsoft Dynamics, orchestrate multifaceted business tasks. While expansive, their intricate nature and steep expenses demand extensive, often cumbersome, customizations [6].

Open-source frameworks furnish developers with pre-existing code segments, expediting development via code recycling. Yet, these necessitate considerable technical acumen and persist with manual coding [9].

In contrast, Metadev merges a lucid graphical interface, code automation, and adaptable capabilities, particularly geared towards ERP systems. This amalgamation optimizes both swiftness and pliability in development, catering to bespoke organizational demands.

To encapsulate, while each method carves its niche, Metadev emerges as a holistic solution amalgamating user-friendliness, auto-code generation, and precise ERP system concentration, marking it as an attractive proposition for software development endeavors.

5. The Solution - Metadev

Metadev, a fusion of "metadata" and "development", is centered on metadata-driven development, emphasizing design over prolonged implementation. Given the agility required in developing expansive ERP systems, Metadev's concept expedites changes while minimizing resource drain. Up next, we delve into the architectural anatomy of Metadev.

5.1 Implementing Layered Architecture

5.1.1 The Problem

Layered architecture, a predominant architectural style, partitions components based on functionality. Ideal implementations, as depicted in Figure 5.1, champion:

1. Consistency: Uniform layering across projects.
2. Browsability: Grouped components ease modification.

Unfortunately, real-world applications often deviate, resulting in convoluted spaghetti code (Figure 5.2).

5.1.2 Solution

Metadev's introduction of agents - Client, Server, and Data - enforces the layered architecture by governing inter-layer communication. These agents obviate spaghetti architecture and preserve layer independence, with configurable communication protocols like REST and GraphQL (Figure 5.3).

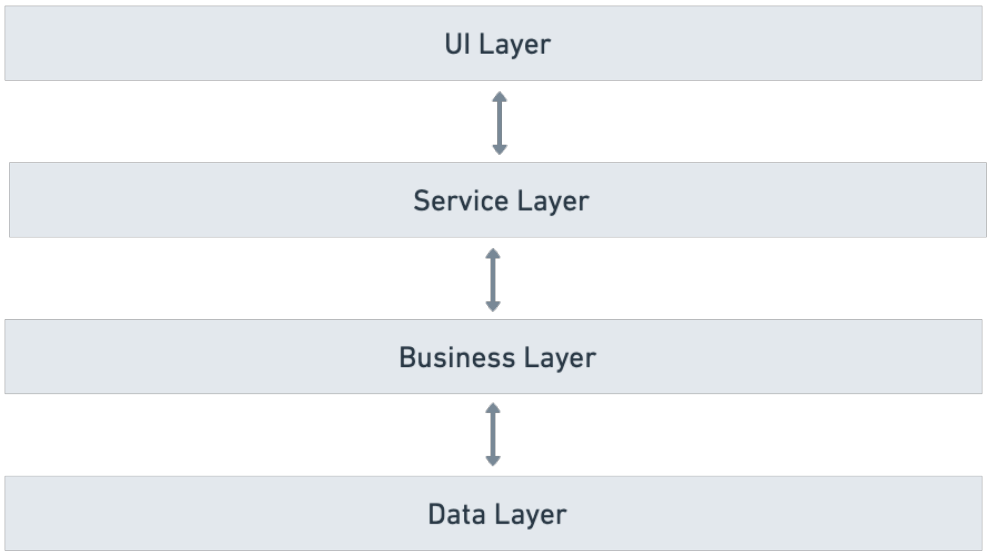


Figure 5.1: Layered Architecture

5.2 Tech Stack Migration

5.2.1 The Problem

As technology races ahead, migrating to new tech stacks becomes cumbersome, fraught with:

1. Application intricacies.
2. Entangled architecture.
3. Business logic entwined with code.
4. Herculean task of reinventing extensive applications.

A case in point is Canvas's codebase (Figure 5.4), where migration means trawling through intertwined business logic in over 100,000 lines.

5.2.2 The Solution

Metadev's metadata-driven technique segregates business specifications from code, relying on generators to interpret metadata and produce code. When switching tech stacks, just a new generator and Metadev's npm package suffice.

Consider an ERP application with over 200 CRUD screens migrating from Angular to React:

Case 1: Traditional Migration - Redoing all 200 screens in React.

Case 2: Metadev - Introducing a React generator for automated screen code generation, supplemented with the mv-react-core library for React-rendering.

Clearly, Metadev provides a leaner, streamlined path for tech stack transitions, especially for expansive ERP applications.

5.3 Detailed Architectural Design

In this section, the framework's detailed architectural design is discussed. The scope of the architecture comprises prompt processing, code generation, stored procedure generation, and UI element rendering. Figure 5.5 provides an overview of the architecture. To achieve the objectives outlined in the previous section, the framework employs the following elements:

1. Metadev JSON Generators
2. Metadev Core Package
3. Metadev Generator
4. Metadev NPM Package
5. Service Agents

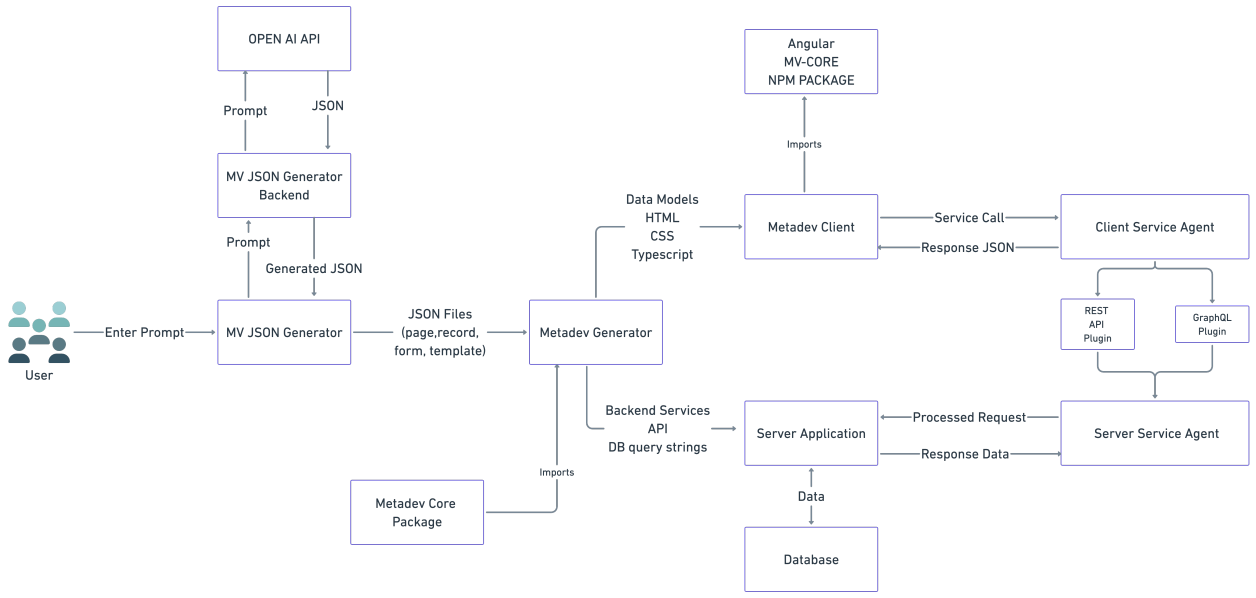


Figure 5.2: Detailed Architecture Diagram

5.3.1 Metadev JSON Generator

Metadev JSON Generator is a user interface for generating metadata from simple prompts and stores them as JSON files. The JSON generator generates the following JSON files:

1. template.json
2. page.json
3. record.json
4. form.json

These JSON files contain the metadata required to generate the end-to-end application code. The purpose and structure of JSON files will be elaborated upon in a later chapter. The process begins with the user entering a prompt containing the page's requirements. For instance, the user may choose to provide a prompt as simple as *students* or as detailed as *A page for students with fields for student name, age, identification number, and place of* *residence.* In both cases, the generator creates the metadata files; however, the first prompt generates a more generic page, with the engine making appropriate assumptions, whereas the second prompt generates a page tailored to the requirements. Figures 5.6 and 5.7 depict simple and detailed prompts in use.

When a user enters a prompt, an HTTP call is made to the MV-JSON-Generator-Backend, and the generated JSON files are returned. The MV-JSON-Generator generates JSON files using the OpenAI chat completion API. The model has been explicitly trained on JSON file-specific example data. Moreover, prompt engineering is performed to ensure the model responds with accurate JSONs.

5.3.2 Metadev Core Package

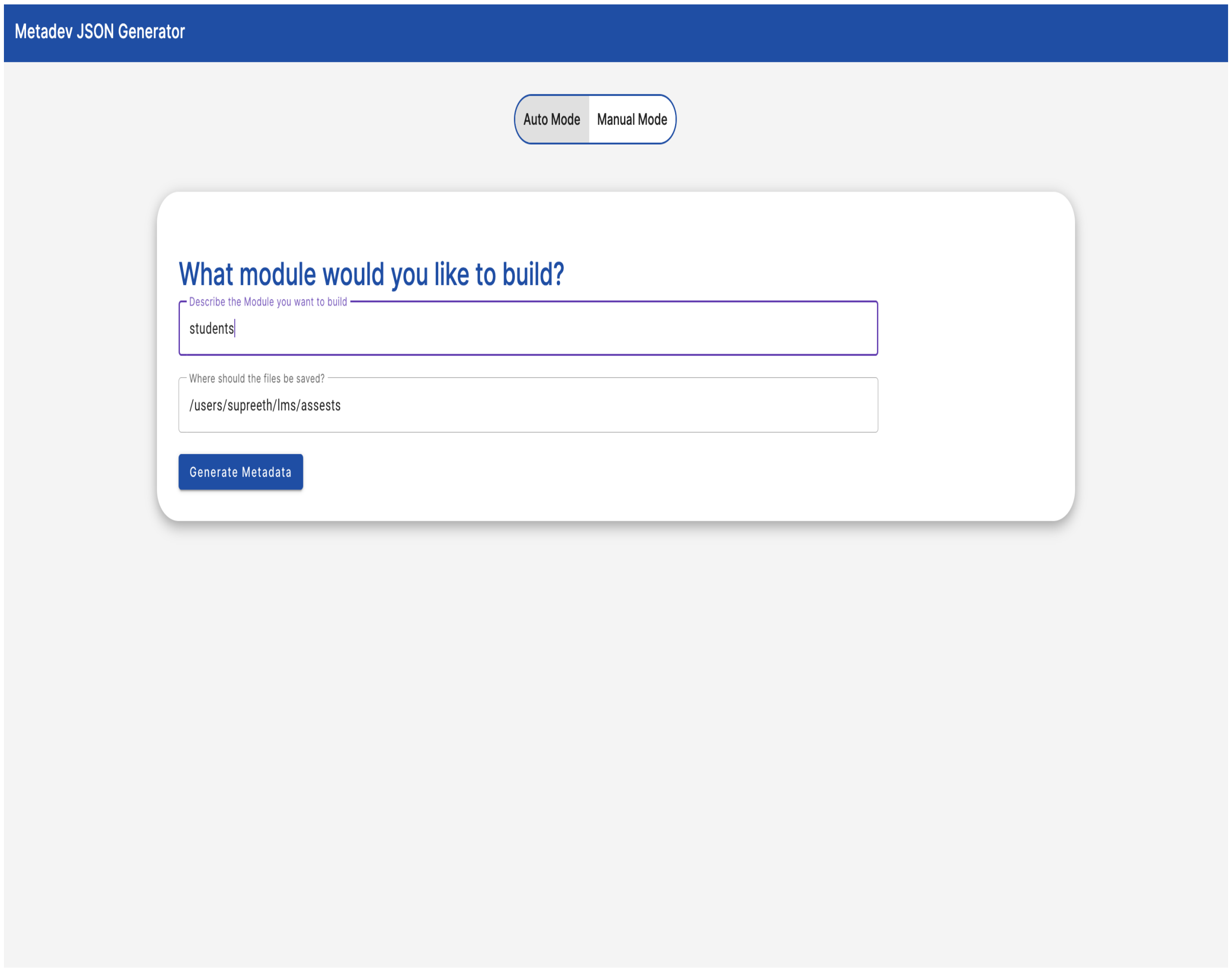
The Metadev core package is the heart of the framework. The package includes the framework datatype definitions, metadata definitions, and the server agent. The server code, produced by the Metadev generator, uses this package to complete all processes.

5.3.3 Metadev Generator

The Metadev generator is tasked with unpacking JSON files and their associated metadata. It produces code for database query strings and endpoints as well as data models, interfaces, and backend services. The bootstrap file acts as the configuration file in which users define the path to the JSON files. Additionally, users can also set up the database by adding the database connection string to the bootstrap file.

5.3.4 Metadev NPM Package

The Metadev NPM package contains the client-side datatype definitions, form functions, form operations, and core client UI elements. The default form functions are:

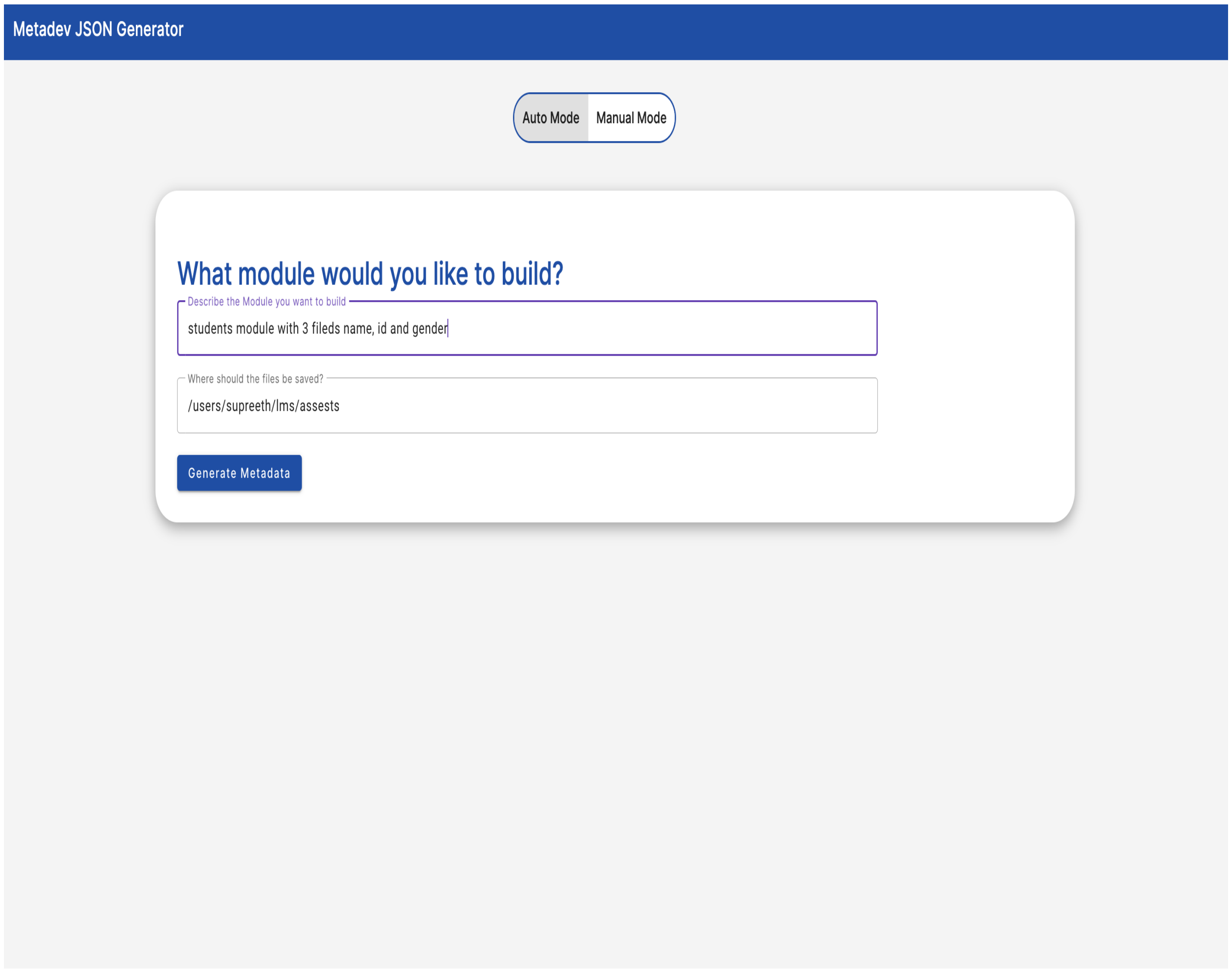


What module would you like to build?

Describe the Module you want to build: students

Where should the files be saved?: /users/supreeth/lms/assests

Figure 5.3: MV-JSON-Generator accepting simple prompt



What module would you like to build?

Describe the Module you want to build: students module with 3 fileds name, id and gender

Where should the files be saved?: /users/supreeth/lms/assests

Figure 5.4: MV-JSON-Generator accepting specific prompt

1. saveAsNew - Create Operation
2. save - Update Operation
3. fetchData - Read Operation
4. delete - Delete Operation
5. setFieldValue - Set a value for a particular field
6. getFieldValue - Get a value for a specific field

All the mentioned functions are asynchronous and return an observable. Users can choose to import the library directly or opt to clone the library and make project-specific design and functionality changes.

5.3.5 Service Agents

In the framework, the communication layer consists of both the client and server service agents. These agents manage communication between the client application and the server application. However, direct communication between client and server applications is not permitted. Instead, the client application interacts with the client service agent, making function calls to request or transmit data, preferably in JSON format. The client service agent is responsible for initiating the required HTTP calls to the server service agent.

Upon receipt of the request, the server service agent activates the appropriate endpoint or service linked to the request. The server then forms a response, which is sent back to the server service agent. The server service agent transmits the response to the client service agent, which unpacks it and forwards it to the client application. This indirect communication procedure ensures the client application remains unaware of the underlying HTTP calls, focusing solely on data handling.

A key benefit of the agents is their ability to seamlessly switch the underlying communication layer. For example, transitioning from REST to GraphQL without adjusting the client or server application code becomes feasible by merely integrating new agents. This adaptability allows applications to effortlessly shift between communication protocols without necessitating extensive code modifications or repetition.

6. Metadev JSON Files

Metadev JSON files are the files which contain the metadata required for the application. The metadata JSONs are of 5 types:

1. application.json
2. *.form.json
3. *.record.json
4. *.page.json
5. *.template.json

All these files would ideally be generated by the MV-JSON-Generator. However, the user can also choose to manually write in all the JSON files or update the generated JSON files based on the requirement. The JSON generator only serves as a tool to assist in writing the JSON files. Each one of these metadata JSON files are discussed in detail in the following sections.

6.1 application.json

The `application.json` file encompasses crucial information specific to the application. This file can be configured to adhere to a multi-tenant architecture, allowing for the specification of the tenant field and the corresponding tenant database table name. The `application.json` file should be ideally used by software architects who understand the requirements of the entire application and/or have a decent understanding of how software is built. In Table 6.1, various attributes and sub-attributes available for utilization within the `application.json` file are discussed.

Table 6.1: Description of attributes in the `application.json` file

Attribute	**Datatype**	**Explanation**	**Example**
name	string	The name of the application.	"metadev-example"
tenantFieldName	string	The name of the tenant field.	"tenantId"
tenantDbName	string	The name of the tenant database.	"tenant_id"
		Lists	
valueLists	Value list objects	Lists containing predefined value lists.	See sub-attributes
keyedLists	Object of keyed list objects	Lists dependent on keys such as phone number	See sub-attributes
		dataTypes	
booleanTypes	Object of boolean type objects	Object containing boolean data types.	See sub-attributes
decimalTypes	Object of decimal type objects	Object containing decimal data types.	See sub-attributes
dateTypes	Object of date type objects	Object containing date data types.	See sub-attributes
integerTypes	Object of integer type objects	Object containing integer data types.	See sub-attributes
textTypes	Object of text type objects	Object containing text data types.	See sub-attributes

Table 6.2: Description of Sub-Attributes

Sub-Attribute	**Value Datatype**	**Example**
Value List Object	array of value and label objects	"accountStatus": [{ "value": "Active", "label": "Active" }]
Keyed List Object	Nested object of key with array of value, label objects	"state": { "130": [{ "value": "Karnataka", "label": "Karnataka" }, { "value": "Tamil Nadu", "label": "Tamil Nadu" },
Boolean type object	object with errorId attribute	"bool": { "errorId": "invalidBool" },
Decimal type object	object with errorId, maxValue, nbrFractions attributes	"internalMarks": { "errorId": "invalidIternalMarks", "maxValue": 100, "nbrFractions": 2 }
Date type object	object with errorId, maxPastDays, maxFutureDays attributes	"date": { "errorId": "invalidDate", "maxPastDays": 73000, "maxFutureDays": 73000 }
Integer type object	object with errorId, minValue, maxValue attributes	"flexibleId": { "errorId": "invalidFlexibleId", "minValue": -1, "maxValue": 9999999999999 }
Text type object	object with errorId, minLength, maxLength and regex attributes	"uniqueId": { "errorId": "invalidUniqueId", "regex": "[1-9][0-9]15", "minLength": 16, "maxLength": 16 }

To provide a comprehensive understanding, Listing 6.1 presents a code snippet exemplifying the structure and content of an `application.json` file.

```
{
    "name": "metadev-example",
    "tenantFieldName": "tenentId",
    "tenantDbName": "tenent_id",
    "dataTypes": {
        "booleanTypes": {
            "bool": {
                "errorId": "invalidBool"
            }
        },
        "decimalTypes": {
            "decimal": {
                "errorId": "invalidDecimal",
                "maxValue": 1000,
                "nbrFractions": 4
            }
        },
        "dateTypes": {
            "date": {
                "errorId": "invalidDate",
                "maxPastDays": 73000,
                "maxFutureDays": 73000
            }
        },
        "integerTypes": {
            "flexibleId": {
                "errorId": "invalidFlexibleId",
                "minValue": -1,
                "maxValue": 9999999999999
            }
        },
        "textTypes": {
            "uniqueId": {
                "errorId": "invalidUniqueId",
                "regex": "[1-9][0-9]{15}",
                "minLength": 16,
                "maxLength": 16
            }
        },
        "timestampTypes": {
            "timestamp": {
                "errorId": "invalidTimestamp"
            }
        }
    },
    "valueLists": {
        "gender": [
            {
                "value": "Male",
                "label": "Male"
            },
            {
                "value": "Female",
                "label": "Female"
            },
            {
                "value": "Others",
                "label": "Others"
            }
        ]
    },
    "keyedLists": {
        "state": {
            "130": [
                {
                    "value": "Karnataka",
                    "label": "Karnataka"
                },
                {
                    "value": "Tamil␣Nadu",
                    "label": "Tamil␣Nadu"
                }
            ]
        }
    }
}
```

Listing 6.1: example application.json

6.2 form.json

`*.form.json files‘ encompass essential metadata pertaining to the client side application. It serves as a means to define the fields, their corresponding types, and labels within the application. Notably, if a modification is required for a field label, it can be accomplished solely by editing the content of the JSON file, without necessitating any changes to the underlying codebase. Additionally, each instance of `*.form.json` must be associated with a distinct `*.record.json` file, establishing a one-to-one relationship between the form and its corresponding record. The `*.form.json` file should ideally used by frontend engineers. In Table 6.2, various attributes and sub-attributes available for utilization within the `*.form.json` files are discussed.

To provide a comprehensive understanding, Listing 6.2 presents a code snippet exemplifying

Table 6.3: Description of attributes in the `*.form.json` file

Attribute	**Data Type**	**Explanation**	**Example**
name	String	Name of the form	orders
recordName	String	Name of the record associated with the form	orders
serveGuests	Boolean	Indicates if non-authenticated requests are to be served	true
operations	Array of Strings	Permitted operations	["Create", "Filter", "Get", "Update"]
controls	Array of Objects	Control details for the table	
Control Details			
name	String	Name of the field to map the form field with the record field	orderId
label	String	Label to be displayed on the UI	Order Name
controlType	Type of UI input	textarea	

the structure and content of an `*.form.json` file. The code snippet is an example for taking in customer details.

```
{
    "name": "customerList",
    "recordName": "customers",
    "serveGuests": "true",
    "operations": ["Create", "Filter", "Update", "Get"],
    "controls": [{
        "name": "customerId",
        "label": "",
        "controlType": "hidden"
    }, {
        "name": "customerName",
            "label": "Customer␣Name",
            "controlType": "input"
        }, {
            "name": "customerEmail",
            "label": "Customer␣Email",
            "controlType": "input"
        },
  {
            "name": "customerPhone",
            "label": "Customer␣Phone",
            "controlType": "input"
        }]
}
```

Listing 6.2: example customers.frm.json

6.3 record.json

`*.record.json` files serve as the specification files for the modules defining the design of the modules. The `*.record.json` files play a pivotal role in establishing a connection between the client application, server application and the associated database. It encompasses essential information regarding the database configuration and serves as a mapping mechanism between the form fields and the corresponding database table. Additionally, `*.record.json` incorporates data validation specifications for the forms on both server and client side. Notably, the relationship between records and forms is defined as one-to-many, allowing multiple forms to utilize the same record.

It is crucial to ensure that the field names in both `*.form.json` files and `*.record.json` files are identical to facilitate accurate mapping between the two entities. This alignment guarantees the proper synchronization and integration of data between the client application and the underlying database. In Table 5.4, various attributes and sub-attributes available for utilization within the `*.record.json` files are discussed.

To provide a comprehensive understanding, Listing 5.3 presents a code snippet exemplifying the structure and content of an `*.form.json` files. The code snippet is an example for taking in customer details.

Table 6.4: Description of Attributes in record.json File

Attribute	**Data Type**	**Explanation**	**Example**
name	String	Name of the record	customers
nameInDb	String	Name of the associated table in the database	customers
operations	Array of Strings	Permitted operations	["Create", "Filter", "Update", "Get"]
fields	Array of Objects	Field details for the record	
Field Details			
name	String	Name of the field	customerId
dbColumnName	String	Corresponding column name in the database	customer_id
dataType	String	Data type of the field	id
fieldType	String	Type of the field	generatedPrimaryKey

```
{
    "name": "customers",
    "nameInDb": "customers",
    "operations": ["Create", "Filter", "Update", "Get"],
    "fields": [{
        "name": "customerId",
        "dbColumnName": "customer_id",
        "dataType": "id",
        "fieldType": "generatedPrimaryKey"
    }, {
        "name": "customerName",
        "dbColumnName": "name",
            "dataType": "name",
            "fieldType": "requiredData"
        }, {
            "name": "customerEmail",
            "dbColumnName": "email",
            "dataType": "email",
            "fieldType": "requiredData"
        }, {
            "name": "customerPhone",
            "dbColumnName": "phone_number",
            "dataType": "phone",
            "fieldType": "optionalData"
        }]
}
```

Listing 6.3: example customers.record.json

6.4 template.json

The purpose of a "*.template.json" files is to serve as a pure UI metadata file, specifically designed for describing the layout of pages that adhere to a particular pattern. It functions as a blueprint for constructing consistent page designs throughout an application. For instance, consider an "add" page that typically consists of a form followed by two buttons: one for saving the data and another for navigating back. By employing a template.json file, a UX architect can establish a definitive structure for the "add" screen. Consequently, if there is a need to modify the design of the "add" screens, such as reducing the number of buttons from two to one, the alteration can be made solely within the template.json file, without requiring any code changes to individual pages. This approach not only streamlines the development process but also ensures standardization across pages, eliminating potential UI errors that may arise. Table 6.5 gives the list of attributes and Listing 6.4 shows an example template.json file.

```
{
    "templateName": "EntryPage",
    "htmlSelector": "app-entry-page",
    "templateType": "form",
    "enableRoutes": true,
    "buttons": [{
        "name": "Save␣Record",
        "action": "Create",
        "buttonType": "Primary",
        "routeOnClick": true
    }, {
        "name": "Navigate␣Back",
        "action": "Navigate",
        "buttonType": "Secondary",
        "routeOnClick": true
    }]
}
```

Listing 6.4: example entry.template.json

6.5 page.json

The `*.page.json` files function as a comprehensive fusion of the `template.json` files and `form.json` files, encompassing the fundamental attributes of an individual page within the application. The primary responsibility of frontend engineers, for creating new pages, revolves around populating the `page.json` files with pertinent metadata. This approach guarantees adherence to UI standards and facilitates time-saving measures.

Table 6.5: JSON Attributes

Attribute	**Data Type**	**Explanation**	**Example**
pageName	string	Name of the page	customerEntry
componentForm	string	Component form for the page	customerList
templateType	string	Type of the template	EntryPage
pageSelector	string	Selector for the page	app-customer-entry
pageRoute	string	Route for the page	customer-entry
navMenuName	string	Name of the navigation menu	Customers
isSavePage	string	Indicates if the page is a save page	True
editRoute	string	Route for editing the page	customer-edit
routeTo	array route to objects	Button Route Definitions	{ "name": "Save Record", "route": "customerList" }
Sub-Attribute: routeTo			
name	string	Name of the route	Save Record
route	string	Route for the route	customerList

7. Empirical Study

An empirical study was conducted to study the time taken to build a simple ERP application by experienced software engineers in their stack of preference vs an engineer experienced in Metadev.

7.1 Setup

For the purpose of an academic report, a group of three highly skilled software engineers, each specialized in different frameworks, and an engineer with expertise in Metadev, were chosen to partake in a challenge. The participants were assigned the same task, and meticulous records of the setup and implementation phases were kept to accurately measure the time taken.

7.2 Task

The task is to build an end-to-end ERP system that includes the UI, server, and database. The ERP system should be capable of displaying, creating, and editing details for the entities and attributes mentioned in Table 7.1.

The application must have the following features:

1. A table to display the data
2. Pre-filled forms for data update
3. Frontend validations
4. Server side validations

Table 7.1: Form Fields

Customer Entity Attributes		
Field Name	Is Required	Shown/Not Shown
Customer ID	Yes	Not Shown
Customer Name	Yes	Shown
Customer Phone Number	No	Shown
Customer Email	Yes	Shown
Order Entity Attributes		
Field Name	Is Required	Shown/Not Shown
Order ID	Yes	Not Shown
Order Reference Number	No	Shown
Order Details	Yes	Shown

5. Database to store and retrieve information

Implementing material design is an optional feature. Please note that the any choice of stack and the actual implementation can be used based on the participant's comfort and experience.

7.3 Result

All 4 participants were able to finish the task and the results are tabulated in Table 7.2

Table 7.2: Comparison of ERP System Development

Participant Number	Stack Chosen	Setup Time (in hours)	Implementation Time (in hours)	Final Time (in hours)
1	Python, SQL, HTML, JS	33	154	187
2	Fast API, SQLite, Svelte	15	182	197
3	NextJs, PostgreSQL	10	191	201
4	Metadev, PostgreSQL	32	8	40

7.4 Summary

Based on the obtained results, it is clear that implementing ERP modules using Metadev is significantly faster compared to other frameworks. However, it should be noted that the setup time for Metadev is relatively longer. This setup time can be reduced significantly with the provision of improved documentation and error reporting. In summary, the empirical study concludes that Metadev is a more time-efficient approach for building ERPs.

8. Limitations

The limitations of Metadev are contingent on the speed and availability of agent development for frameworks like ReactJs, VueJs, NextJs, etc. While users can craft their own agents, the framework's full potential manifests when multiple agents are accessible. The capacity of Metadev to smoothly transition projects not originally built with it is also unconfirmed. These constraints are pivotal when gauging Metadev's efficiency and relevance.

9. Future Work

The existing Metadev version is a metadata-driven development proof of concept. It houses an Angular frontend package, a SOA communication layer agent, and a Java generator. Future pursuits include adding frontend packages for ReactJS, VueJS, etc., and building new communication layer agents for REST and GraphQL. Backend generators for languages like Python, C#, etc., can also be introduced. Additionally, considering the rise of mobile devices, supporting native apps for iOS and Android is vital. Envisioning a UI akin to Android Studio or Xcode could preclude manual JSON metadata writing. The framework's zenith is generating diverse native application versions across multiple tech stacks using identical JSON files, which will truly segregate the development process.

10. Conclusion

Traditional ERP system development methods encounter various hindrances. Manual coding, IDEs, ERP systems (both commercial and open-source), and low-code/no-code platforms have restrictions that deter development agility and code understanding. Transitioning apps to adapt to modern technologies is both prolonged and pricey.

Metadev, introduced in this academic report, addresses these issues. It's an open-source tool that produces frontend and backend ERP application code, balancing easy interfaces and customization. Metadev fosters efficient development, simplifies procedures, and engages non-tech users. It complements standard ERP application's N-tier architectural patterns and overcomes previous solutions' limitations.

After contrasting with alternate solutions, this report extols Metadev's distinct attributes and its proficiency in refining software development methods. It caters to different entities, from NGOs to tech giants.

Conclusively, Metadev emerges as a potent solution to traditional ERP development issues. Its frontend and backend code generation capabilities, emphasis on customization and accessibility, earmark it as indispensable for both technical and non-technical users. With Metadev, entities can elevate development productivity, champion best practices, and realize bespoke ERP solutions.

References

[1] Gartner, "Market share analysis: Erp software, worldwide, 2020." https://www.gartner.com/en/documents/4000842, 2020.

[2] P. Consulting, "Average erp implementation time." https://www.panorama-consulting.com/average-erp-implementation-time/, 2021.

[3] NetSuite, "Erp benefits." https://www.netsuite.com/portal/resource/articles/erp/erp-benefits.shtml, 2023.

[4] Y. Liu and G. Ruhe, "Visual programming environments: A survey," *ACM Computing Surveys*, vol. 40, no. 4, pp. 1–52, 2008.

[5] A. E. Hassan and P. Jalote, "The impact of coding standards on software quality: A case study," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 981–993, 2001.

[6] B. W. Boehm and P. J. Papaccio, "Understanding and controlling software costs," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1412–1421, 1988.

[7] CloudZero, "Cloud migration 101: The complete guide to migrating to the cloud," May 2023.

[8] Y. Chen, L. Wang, and Y. Zhang, "A survey on code generation for enterprise resource planning systems," *Journal of Systems and Software*, vol. 131, pp. 128–145, 2017.

[9] Y. Zhang, Y. Chen, and L. Wang, "A survey on low-code/no-code platforms," *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–36, 2019.