Executive Summary of Wargaming_ERP Program

I am currently using MySQLWorkbench to create my DB. I have tried to keep it normalized as best as possible. The overall purpose of this database is to be able to take the source data from onepagerules.com Army Forge, as well as the Warhammer 40k 10th edition wargaming rules sets available from wahapedia.ru, create army lists that are game legal (which includes WAY more game rules knowledge to implement into the app than I would have imagined going in). I currently have imported two different datasets from two disparate sources. The Wahapedia downloads were in a semi structured set of CSV files. The Army Forge dataset was contained in a large JSON that required a significant amount of parsing and troubleshooting to even get imported correctly. I have tried to normalize this data as much as possible to 3NF, but there are plenty of tables that use composite keys and other things that keep it from being truly 3NF, mostly by virtue of the structure of the data involved. I am currently working on that game logic, which is the part that I really feel is outside the scope of the project.

Eventually, I plan to add a third data source, which is a JSON of metadata regarding myminifactory.com 3D model information. The final goal of this app is to be able to cross reference the units from the rules of these two different wargaming systems, build game legal army rosters, and recommend proxy units that can be 3D printed. There are some more lofty goals to include, among many, comparison against official Games Workshop retail prices that will probably require a 4th data dump, so that is a bridge to be crossed at a later date.

I am taking a lot of my inspiration from current options already in the marketplace like Army Forge, New Recruit, and Wahapedia. I am hoping to mimic a lot of their functionality in the army roster side, but adding the extra bit about the 3d model file information.

For software, I am using MySQLWorkbench for creating and hosting the DB, Streamlit and Python for the actual app operational code. That code is currently broken into 4 operational pillars, including the app.py , which handles the overall app launching, database_utils.py that contains the login information for the MySQL DB, opr_builder.py that handles building armies under the OPR ruleset, and w40K_builder that handles building armies under the 40K rule set. I have created a Github repository for revision control, because I was getting lost in the weeds of all the code changes and feature implementation breaking two features every time I got one working.

Eventually, I would like to be able to actually use this program in my hobby to make armies from scratch to play in wargames. I have learned an awful lot (and not nearly as much as I would like to know), about database creation and management, software development, change control, UI Design, Python as a scripting language, and a slew of other subjects along the way, like how to create and parse JSON and CSV files and reverse engineer DB schema from that data.

I have created (so far) 43 Tables, 10 Views, and 2 routines. My web app is currently able to create new army rosters for each rule set, all units, filter the selection list by army or detachment, display unit “Data Cards” that show individual unit information, allow selections of optional wargear with persistence for in roster selections. I am currently working on the features that will make the list building more game legal and intuitive, but that is a lot of tedious slow going.

Many of the features that I have implemented have required stacked and nested joins, as well as the creation of temporary and additional tables to contain data for views and the different normalized data sets from the original data dumps. I have built a readme and a Data Dictionary about the DB schema and structure, including table names, column names, data types, etc into my Github for help with building queries and troubleshooting issues in the DB, the app, and the integration between the two. I am including the dump of my DB tables with data, and the stored views. I am not sure what else I could include to show that the depth of this project execution far exceeds anything that the syllabus for this class requires. If I am mistaken in this assessment, please let me know what parts I have missed. I think I have hit all the major pillars of this course and now I am in the feature development and refinement stage, which I feel is probably beyond the scope of this class.

Thanks,
John Mills

README.md

🛡️ Wargaming ERP Technical Documentation

Version: 1.0.0 (Modular Release)

Game Systems: Warhammer 40,000 (10th Ed), OPR (Grimdark Future/Age of Fantasy)

🏗️ 1. Architecture Overview

The application follows a Modular Streamlit Architecture. Logic is separated by game system to prevent cross-contamination of rules and data structures.

app.py: The Traffic Controller. Handles list creation and routes the user to the correct builder module based on the game_system flag.

database_utils.py: The Foundation. Centralizes the mysql-connector logic.

w40k_builder.py: The 10th Edition sandbox. Handles Detachments, 6-metric stat bars, and Enhancement logic.

opr_builder.py: The OPR engine. Handles "Replace" wargear logic, strikethrough rendering, and upgrade sets.

🗄️ 2. Database Schema (Standardized)

The database is hosted on MySQL 8.0. It bridges flat Wahapedia data with hierarchical OPR JSON data.

Core Management Tables

play_armylists: The header. Stores list_name, point_limit, faction_primary, faction_secondary, and waha_detachment_id.

play_armylist_entries: The junction. Links a list_id to a unit_id.

play_armylist_upgrades: Tracks OPR-specific upgrade selections.

play_armylist_enhancements: Tracks 40K-specific Character enhancements.

System Mapping Views

view_master_picker: The "Search Engine." A UNION ALL view that standardizes Unit Name, Faction, and Points across all games so the UI search bar works universally.

view_40k_datasheet_complete: Flattens 40K stats (M, T, Sv, W, etc.) into a single row for the UI Stat Bar.

⚙️ 3. Core Logic & Stored Procedures

We use Stored Procedures to handle heavy logic inside the database, keeping the Python code lean.
GetArmyRoster(list_id)
The "Brain" of the Roster. It calculates:
Base Cost of the unit.
Upgrade Cost (OPR) by joining play_armylist_upgrades.
Enhancement Cost (40K) by joining play_armylist_enhancements.
Multiplication of (Base + Upgrades) * Quantity.
Stats: Conditionally returns 40K stats or OPR stats based on the list's game_system.
🎨 4. UI Design Standards
Unit Detail Pop-ups (@st.dialog): Triggered via the 👁️ icon.
OPR: Features "Active Weapons" vs "Upgrades" tabs with strikethrough logic (🚫) for replaced gear.
40K: Features a 6-column metric bar and "Special Rules" pulled from the waha_abilities table.
Cascading Filters: List creation and Library search use a 3-tier cascade: System -> Setting -> Faction.
🛠️ 5. Maintenance & Troubleshooting
Adding New OPR Data: Use the OPR_JSON_analyzer.py script. It automatically populates opr_units, opr_unit_upgrades, and maps the opr_army_settings.
Metadata Locks: If the app hangs, use SHOW PROCESSLIST and KILL [ID] in Workbench to clear stuck metadata locks on Stored Procedures.
Safe Updates: Run SET SQL_SAFE_UPDATES = 0 when performing mass updates on play_armylists or waha_factions.
🚀 6. Future Roadmap
40K Enhancement Toggle: Adding the checkbox logic to the 40K Details tab.
40K Wargear Swaps: Implementing the replacement/strikethrough logic for 40K weapon options.
Validation Engine: "Rule of 3" checks and OPR "1 Hero per 500pts" enforcement.

🏗️ 40K Wargear Logic Strategy
In 40K 10th Edition, wargear is mostly free, but the logic is "Replace A with B."
Storage: We'll use a new bridge table play_armylist_wargear_selections.
UI: Inside the "Weapons" tab, we'll list the options.
Visuals: If you select "Replace Bolt Pistol with Plasma Pistol," we'll strike through the Bolt Pistol in the weapon table, just like we did for OPR.

DataDictionary.md
📚 Data Dictionary: Wargaming ERP
Database: wargaming_erp | Engine: InnoDB | Charset: utf8mb4
🛡️ 1. List Management (Core App Logic)
These tables store the lists you create and the units you add to them.

Table	Column	Type	Description
play_armylists	list_id	INT (AI)	Primary Key. Unique ID for each army list.
	list_name	VARCHAR(100)	The user-defined name of the roster.

game_system ENUM 'OPR' or '40K_10E'. Controls UI logic.
point_limit INT The points ceiling for the roster.
faction_primary VARCHAR(100) Main army book/faction.
faction_secondary VARCHAR(100) Allied army book/faction (Optional).
waha_detachment_id VARCHAR(50) FK to waha_detachments. Sets 40K ruleset.
play_armylist_entries entry_id INT (AI) Primary Key. Represents a unit "instance" in a list.
list_id INT FK to play_armylists.
unit_id VARCHAR(50) Links to waha_datasheet_id or opr_unit_id.
quantity INT Number of models/units in this entry.
🎲 2. OPR Data Structures
Populated via the OPR_JSON_analyzer.py deep-dive.
Table Column Type Description
opr_units opr_unit_id VARCHAR(50) PK. Unique ID from OPR JSON.
name VARCHAR(100) Unit name (e.g., "Hive Lord").
army VARCHAR(100) Army book name.
base_cost INT Points before upgrades.
quality INT Quality stat (e.g., 3 means 3+).
defense INT Defense stat (e.g., 2 means 2+).
wounds INT Health of the unit.
opr_unit_upgrades id INT (AI) PK. Unique ID for the upgrade option.
unit_id VARCHAR(50) FK to opr_units.
section_label VARCHAR(255) Category (e.g., "Replace Shredder Cannon").
option_label VARCHAR(255) The choice (e.g., "Acid Cannon").
cost INT Point cost of this upgrade.
🦅 3. 40K 10th Edition Data
Standardized Wahapedia architecture.
Table Column Type Description
waha_datasheets waha_datasheet_id VARCHAR(50) PK. Unique Wahapedia ID.
name VARCHAR(100) Unit name.
faction_id VARCHAR(50) FK to waha_factions (e.g., 'ORK').
points_cost INT Current 10E points per model/unit.
movement VARCHAR(10) M stat (e.g., '6"').
toughness INT T stat.
waha_detachments id VARCHAR(50) PK. Unique ID for the sub-faction rules.
faction_id VARCHAR(50) Links to the parent faction.
name VARCHAR(100) Detachment name (e.g., "War Horde").
🛠️ 4. Upgrade & Enhancement Junctions
Tracks what the user "bought" for their units.
Table Column Type Description
play_armylist_upgrades selection_id INT (AI) PK. Tracks OPR gear swaps.
entry_id INT FK to play_armylist_entries.
upgrade_id INT FK to opr_unit_upgrades.
play_armylist_enhancements selection_id INT (AI) PK. Tracks 40K Character relics.

entry_id INT FK to play_armylist_entries.
enhancement_id VARCHAR(50) FK to waha_enhancements.

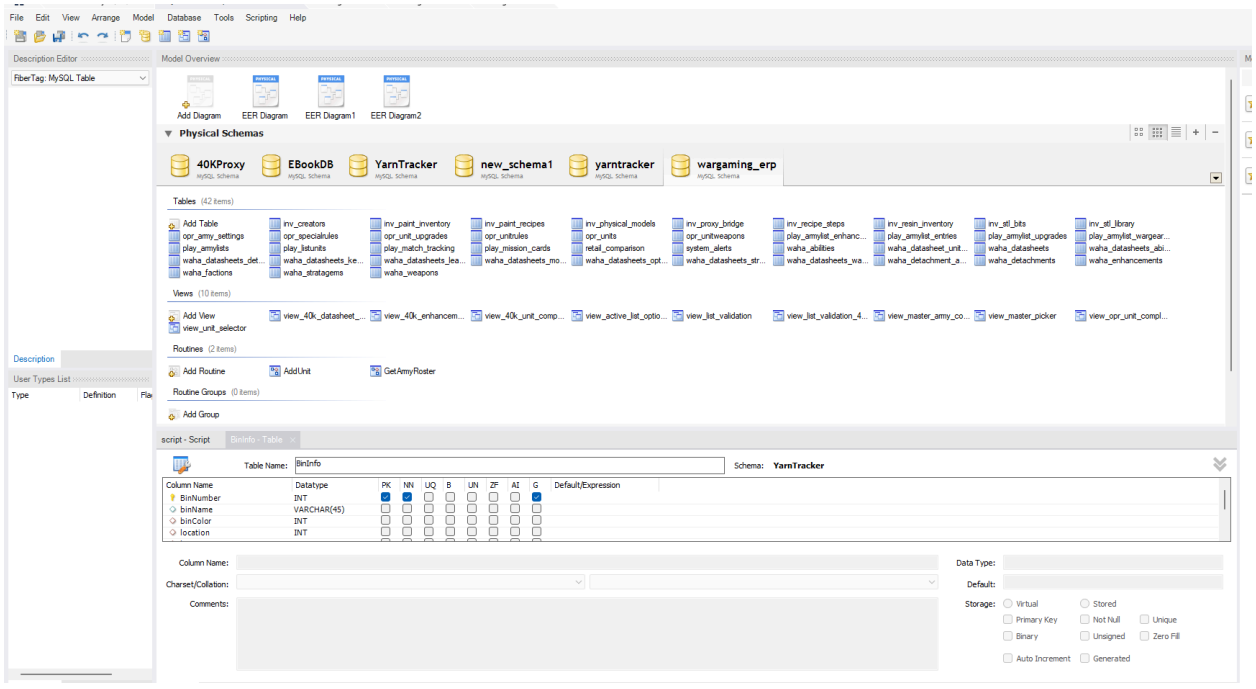
💡 Pro-Tip
You can keep your schema documented and up-to-date automatically by running this SQL query anytime you make changes:

```
SELECT
   TABLE_NAME, COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
FROM
   INFORMATION_SCHEMA.COLUMNS
WHERE
   TABLE_SCHEMA = 'wargaming_erp'
ORDER BY
   TABLE_NAME, ORDINAL_POSITION;
```

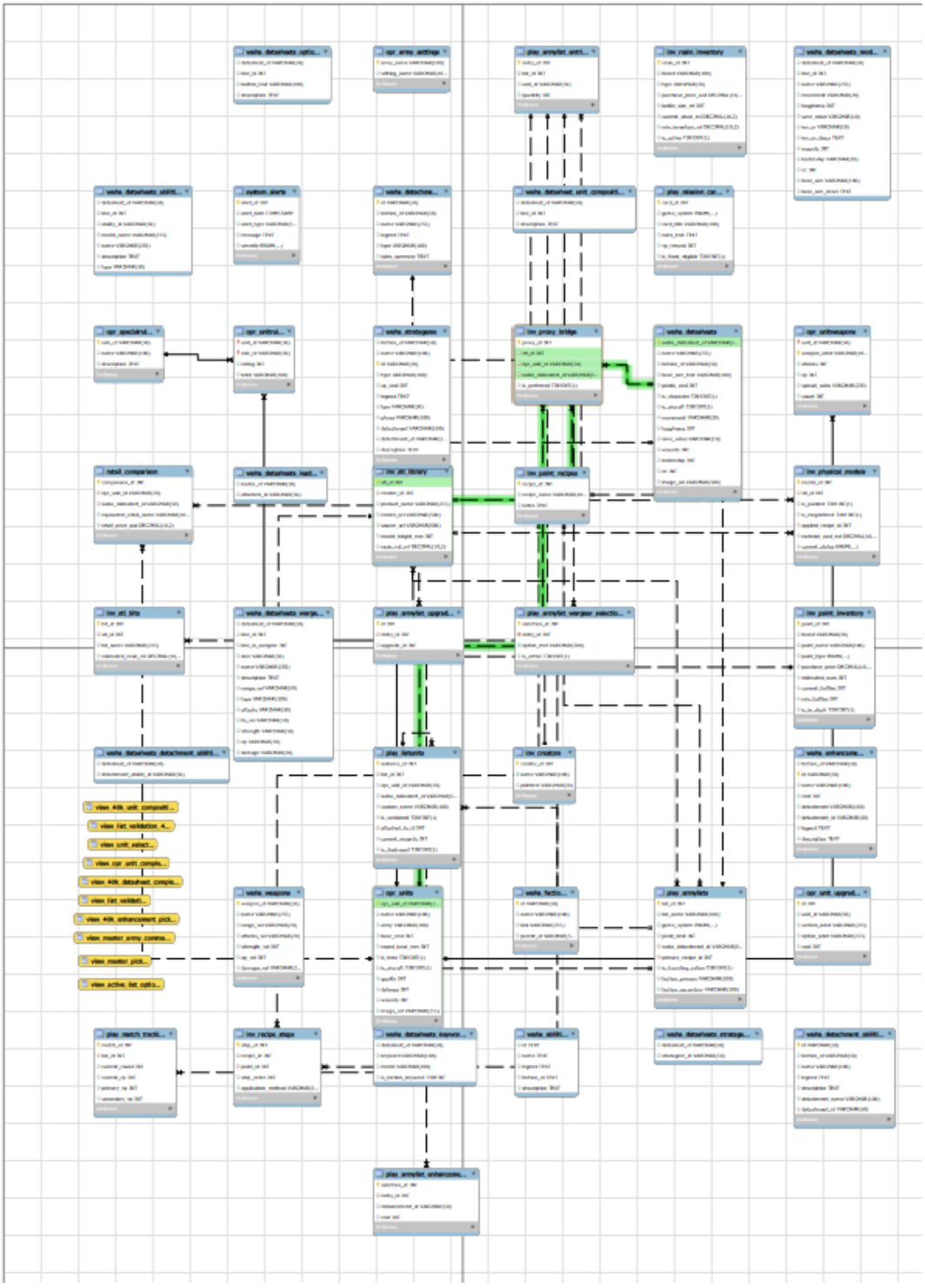
Full Table and column layout:

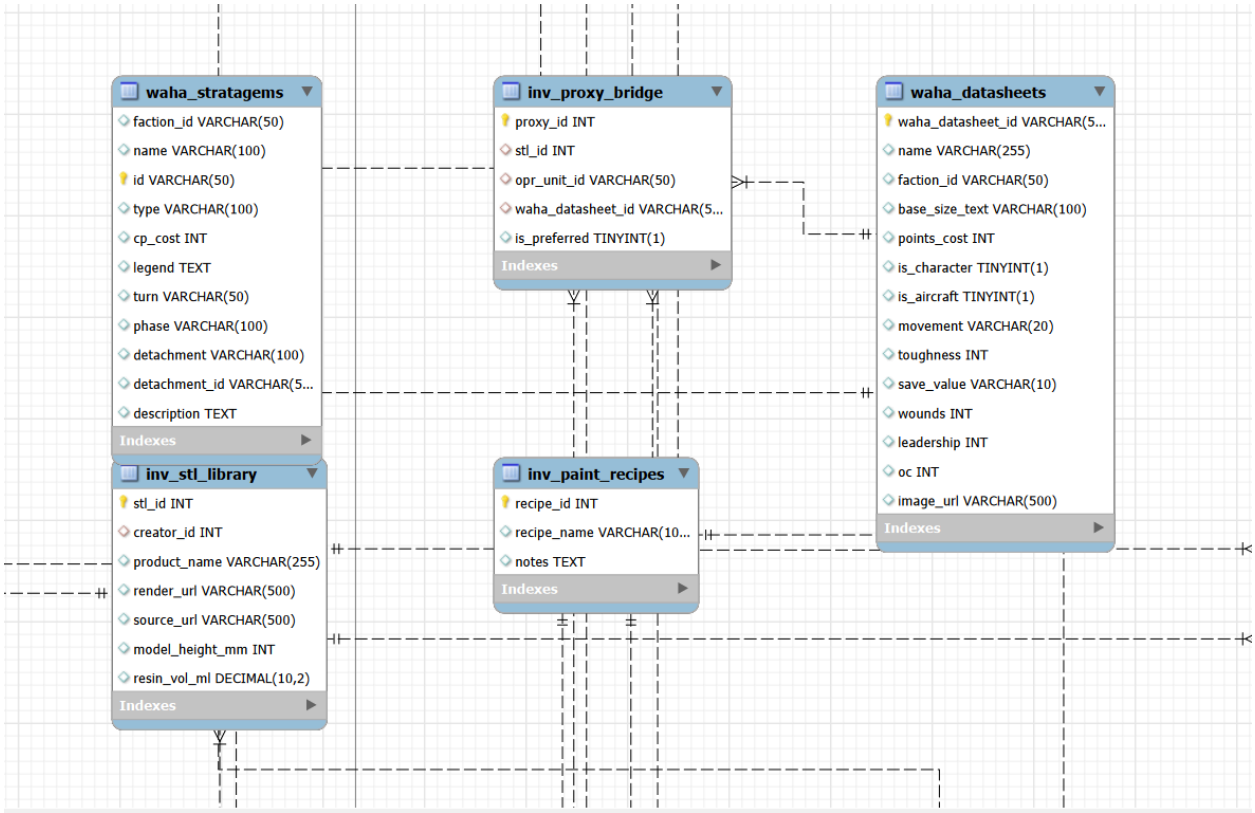
Screenshots

MySQLWorkbench Dashboard

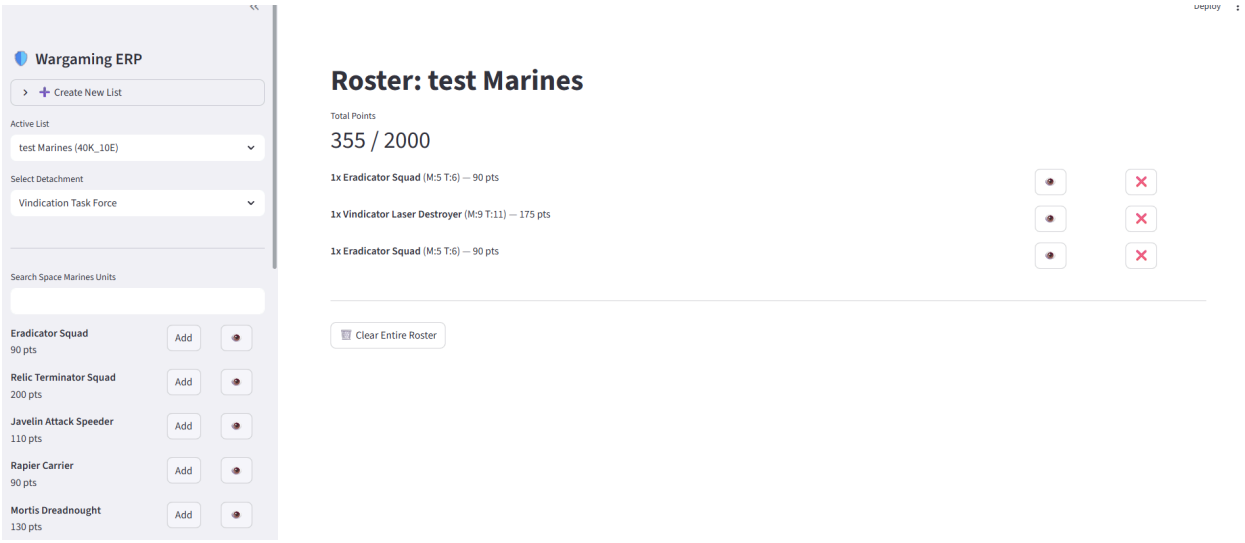


EER Diagram for Wargaming ERP

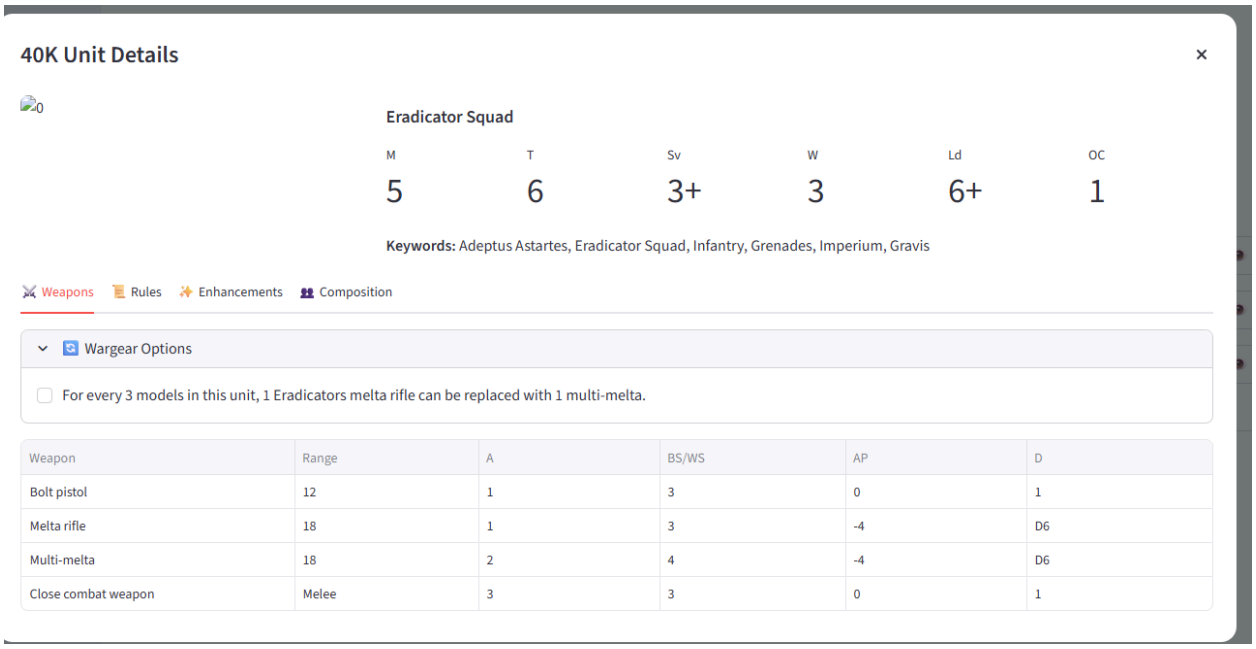



waha_stratagems
faction_id VARCHAR(50)
name VARCHAR(100)
id VARCHAR(50)
type VARCHAR(100)
cp_cost INT
legend TEXT
turn VARCHAR(50)
phase VARCHAR(100)
detachment VARCHAR(100)
detachment_id VARCHAR(5...
description TEXT
Indexes
inv_proxy_bridge
proxy_id INT
stl_id INT
opr_unit_id VARCHAR(50)
waha_datasheet_id VARCHAR(5...
is_preferred TINYINT(1)
Indexes
waha_datasheets
waha_datasheet_id VARCHAR(5...
name VARCHAR(255)
faction_id VARCHAR(50)
base_size_text VARCHAR(100)
points_cost INT
is_character TINYINT(1)
is_aircraft TINYINT(1)
movement VARCHAR(20)
toughness INT
save_value VARCHAR(10)
wounds INT
leadership INT
oc INT
image_url VARCHAR(500)
Indexes
inv_stl_library
stl_id INT
creator_id INT
product_name VARCHAR(255)
render_url VARCHAR(500)
source_url VARCHAR(500)
model_height_mm INT
resin_vol_ml DECIMAL(10,2)
Indexes
inv_paint_recipes
recipe_id INT
recipe_name VARCHAR(10...
notes TEXT
Indexes

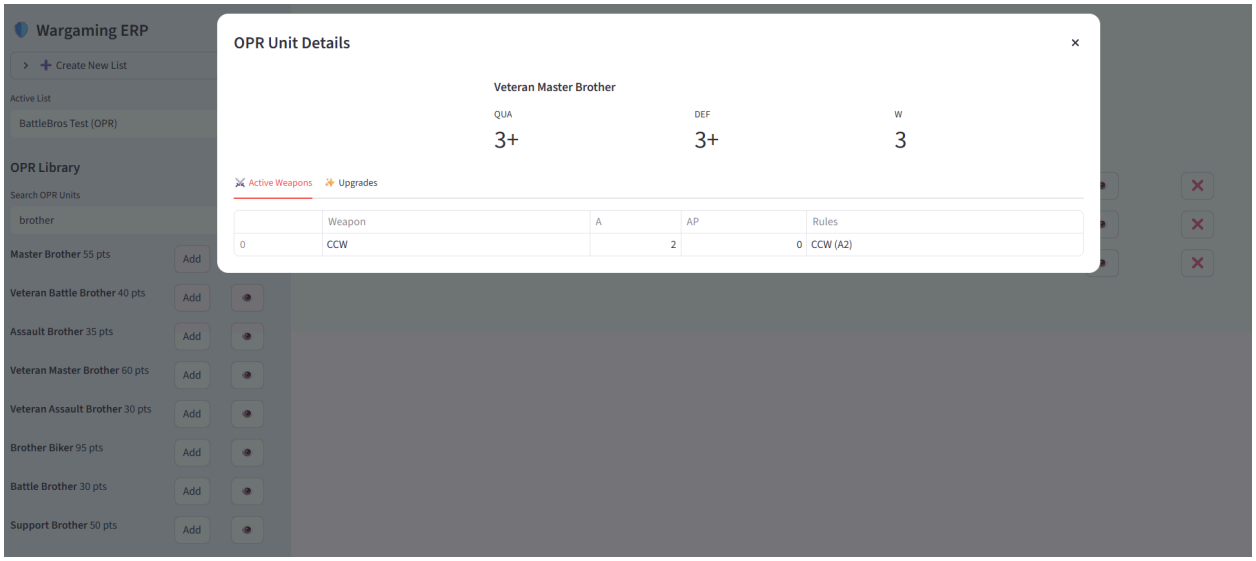
Screen shots from the web app
Army builder screen (40K)



40K Data Card view



OPR Data Card View



OPR Enhancement Picker