# Plan budowy aplikacji do zarządzania listami armii i wyposażenia

## 1. Architektura aplikacji

**Warstwa backend (serwerowa):** Aplikacja zostanie zbudowana w oparciu o język Python – proponowany jest framework webowy (np. Flask lub Django) do obsługi logiki serwera i komunikacji HTTP. Backend będzie odpowiadał za zarządzanie danymi (armie, jednostki, bronie, listy armii itp.), logikę biznesową (np. obliczanie kosztów punktowych zgodnie z zasadami gry) oraz uwierzytelnianie użytkowników. Logika zostanie podzielona na moduły odpowiadające głównym funkcjonalnościom: - **Moduł kont użytkowników:** obsługuje rejestrację/logowanie, profile użytkowników i uprawnienia. Ponieważ aplikacja ma działać w zaufanym środowisku, mechanizm logowania może być uproszczony (np. podstawowe loginy bez zaawansowanej weryfikacji)[[1]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Mo%C5%BCliwo%C5%9B%C4%87%20zalogowania%20si%C4%99,link%20do%20aplikacji%20jest%20zaufany), a niezalogowani użytkownicy korzystają ze wspólnego profilu domyślnego[[1]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Mo%C5%BCliwo%C5%9B%C4%87%20zalogowania%20si%C4%99,link%20do%20aplikacji%20jest%20zaufany). - **Moduł zbrojowni (broni):** obsługuje tworzenie, edycję, usuwanie i wyświetlanie wpisów broni. Przechowuje informacje o każdej broni (nazwa, zasięg, liczba ataków, penetracja pancerza (AP), specjalne cechy itd.) oraz jej koszt. Logika biznesowa obejmie obliczanie kosztu broni na podstawie statystyk i cech specjalnych (zgodnie z formułami z zasad). - **Moduł armii i jednostek:** umożliwia definiowanie armii (frakcji) oraz przypisanych do nich jednostek. Każda jednostka ma nazwę, statystyki (Jakość, Obrona, Wytrzymałość), listę zdolności/cech oraz domyślną broń[[2]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Armie%3A). Ten moduł obsłuży dodawanie nowych armii, kopiowanie istniejących oraz tworzenie wariantów armii (dziedziczenie jednostek)[[3]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Armie%20mog%C4%85%20by%C4%87%20tworzone%20od,jednostki%20z%20odno%C5%9Bnikami%20do%20bazowych). Podobnie na poziomie jednostki – będzie można edytować domyślną broń lub cechy jednostki, z zachowaniem relacji do bazowych obiektów, jeśli jednostka pochodzi z armii bazowej (szczegóły mechanizmu dziedziczenia poniżej). - **Moduł list armii (rozpisek):** odpowiada za tworzenie konkretnych list armijnych do gry. Użytkownik wybiera armię bazową dla rozpiski (lub kilka armii, jeśli dozwolone są sojusze) i dodaje jednostki z tej armii do listy[[4]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Przy%20tworzeniu%20sugerowane%20jest%20wybranie,armii). Funkcjonalność obejmie możliwość modyfikacji liczebności oddziałów oraz ich wyposażenia na poziomie listy. Backend przelicza koszt punktowy każdej jednostki (oraz całej listy) dynamicznie i weryfikuje ograniczenia (np. limit punktów, unikalność jednostek jeśli dotyczy). Na tym etapie dostępne będą również funkcje eksportu – wygenerowanie gotowej listy w formacie PDF lub Excel. - **Moduł eksportu i generowania raportów:** generuje podgląd listy armii w formatach przyjaznych do wydruku oraz odpowiada za utworzenie pliku PDF lub XLS(X). Może wykorzystywać szablony HTML/CSS (np. układ „ramka” lub prosty listowy) do zrenderowania listy, która następnie zostanie przekształcona do PDF. Eksport do Excela będzie opcjonalny – wygenerowanie pliku z arkuszem zawierającym tabelaryczne zestawienie armii[[5]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Eksport%3A). - **Moduł logiki zasad gry:** zbiór funkcji/klas odpowiedzialnych za zgodność z zasadami *OPR*. Będą tu zaimplementowane m.in. funkcje do obliczania kosztu bazowego modelu, kosztu broni i zdolności, zgodnie z formułami z dokumentu zasad[[6]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Koszt%20modelu%20jest%20sum%C4%85%20kosztu,kosztu%20broni%20i%20kosztu%20zdolno%C5%9Bci)[[7]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Koszt%20zdolno%C5%9Bci%20pasywnych%20nale%C5%BCy%20zap%C5%82aci%C4%87,za%20ka%C5%BCdy%20punkt%20wytrzyma%C5%82o%C5%9Bci). Przykładowo, koszt modelu = koszt bazowy + koszt broni + koszt wszystkich zdolności[[6]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Koszt%20modelu%20jest%20sum%C4%85%20kosztu,kosztu%20broni%20i%20kosztu%20zdolno%C5%9Bci), gdzie koszt bazowy wylicza się ze statystyk (np. 5 \* modyfikator jakości \* modyfikator obrony \* modyfikator wytrzymałości[[6]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Koszt%20modelu%20jest%20sum%C4%85%20kosztu,kosztu%20broni%20i%20kosztu%20zdolno%C5%9Bci)), a koszt zdolności pasywnych mnożony jest przez liczbę punktów Wytrzymałości modelu[[7]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Koszt%20zdolno%C5%9Bci%20pasywnych%20nale%C5%BCy%20zap%C5%82aci%C4%87,za%20ka%C5%BCdy%20punkt%20wytrzyma%C5%82o%C5%9Bci). Te reguły zostaną odwzorowane w kodzie, aby aplikacja automatycznie kalkulowała koszty zgodnie z zasadami. Dodatkowo moduł ten będzie zawierał słownik dostępnych zdolności (podzielonych na zdolności jednostek i broni), ich opisów oraz zasad wpływu na koszt[[8]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Lista%20zdolno%C5%9Bci%20wraz%20z%20opisami,i%20zasadami%20obliczania). Dzięki temu przy edycji jednostki lub broni system może automatycznie uwzględniać wpływ dodania danej zdolności na koszt punktowy (np. *Strach(X)* zwiększa koszt o 0,75 \* X punktów[[9]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Transport,Lataj%C4%85cy%2C%201%2C25%20Szybki%2FZwinny%2C%201%20pozosta%C5%82e), a aura zdolności działa jak wykupienie tej zdolności dla 8 punktów wytrzymałości itp.).

**Warstwa frontend (interfejs użytkownika):** Frontend zostanie zrealizowany jako responsywna aplikacja webowa dostępna przez przeglądarkę. Planowane jest użycie szablonów HTML (np. Jinja2 w Flasku lub Django Templates) wraz z arkuszami stylów CSS (np. framework Bootstrap) do zapewnienia czytelnego i skalowalnego interfejsu na różnych urządzeniach (monitor, laptop, tablet, smartfon)[[10]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Skalowalno%C5%9B%C4%87%20do%20wy%C5%9Bwietlania%20na%20ekranie,panoramicznym%2C%20laptopie%2C%20tablecie%2C%20telefonie). Struktura interfejsu będzie odpowiadać głównym modułom: - **Strona główna (dashboard):** wyświetla podsumowanie i nawigację: listę utworzonych rozpisek (list armii) z opcją utworzenia nowej, listę zdefiniowanych armii (również z opcją utworzenia nowej) oraz odnośniki do innych modułów[[11]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Strona%20g%C5%82%C3%B3wna%3A). Stąd użytkownik może przejść do „Zbrojowni” (listy broni), edytora armii, edycji wariantów zasad (jeśli zaimplementowane) oraz do podglądu/exportu konkretnej listy. - **Zbrojownia (ekran broni):** tabela z wszystkimi dostępnymi broniami. Kolumny mogą obejmować m.in.: nazwę broni, zasięg, liczbę ataków, wartość AP (przebicie pancerza), specjalne cechy, koszt punktowy. U góry strony – przycisk „Dodaj nową broń”. Po kliknięciu dostępny będzie formularz dodawania/edycji broni, w którym użytkownik uzupełni wszystkie ww. pola. Edycja/usuwanie istniejących wpisów również będzie możliwa (np. poprzez ikonki „Edytuj”/„Usuń” przy każdej pozycji)[[12]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Lista%20broni%20%E2%80%93%20uk%C5%82ad%20podobny,jak%20we%20wzorcowym%20Excelu). Układ listy broni będzie wzorowany na istniejącym arkuszu Excel udostępnionym przez użytkownika, co ułatwi orientację w danych[[13]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Zbrojownia%3A). - **Edytor armii (ekran armii):** lista wszystkich zdefiniowanych armii. Dla każdej armii wyświetlana będzie nazwa frakcji oraz liczba jednostek zdefiniowanych w jej ramach. Przy armii pojawią się opcje „Edytuj”, „Usuń”, a na górze „Utwórz nową armię”. Po wybraniu konkretnej armii przechodzi się do widoku edycji armii: u góry nagłówek z nazwą armii i ewentualnym wyborem wariantu zasad (jeśli różne warianty zmieniają koszty czy dostępne zdolności). Niżej – lista jednostek należących do tej armii[[2]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Armie%3A). Dla każdej jednostki pokazane będą podstawowe atrybuty (Jakość, Obrona, Wytrzymałość), jej zdolności oraz domyślna broń. Dostępne akcje to dodanie nowej jednostki, edycja istniejącej (np. zmiana atrybutów, dodanie/usunięcie zdolności lub zmiana domyślnej broni) oraz usunięcie jednostki. Podczas dodawania lub edycji jednostki, jeśli chcemy przypisać jej broń, interfejs zaoferuje wybór spośród istniejących wpisów broni: najpierw broń domyślna (jeśli jednostka kopiowana z innej armii), następnie lista broni należących do danej armii, a dalej ogólna zbrojownia globalna[[14]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Lista%20modeli%20w%20armii%3A%20nazwa%2C,podstawowe%20atrybuty%2C%20zdolno%C5%9Bci%2C%20domy%C5%9Blna%20bro%C5%84). Dzięki temu użytkownik szybko znajdzie odpowiednią broń lub doda nową w locie. Funkcja tworzenia armii jako wariantu innej armii zostanie zrealizowana poprzez mechanizm dziedziczenia – np. kliknięcie „Utwórz wariant armii” pozwoli wybrać istniejącą armię bazową; nowa armia zostanie w bazie powiązana z tą armią jako rodzicem, a wszystkie jej jednostki będą początkowo skopiowane z jednostek bazowych (ze wskazaniem referencji)[[3]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Armie%20mog%C4%85%20by%C4%87%20tworzone%20od,jednostki%20z%20odno%C5%9Bnikami%20do%20bazowych). - **Kreator listy (ekran rozpiski):** umożliwia złożenie listy armii do rozgrywki. Użytkownik na początku wybiera, dla jakiej armii (lub armii) tworzy rozpiskę[[4]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Przy%20tworzeniu%20sugerowane%20jest%20wybranie,armii). Następnie widzi nagłówek z nazwą armii, polem do wpisania limitu punktów oraz sumą punktów aktualnie wydanych w liście. Pod nagłówkiem znajduje się lista jednostek dostępnych w danej armii – przy każdej z nich opcja „Dodaj do listy”. Możliwe jest także dodanie jednostki spoza armii (np. najemników lub sojuszników) albo całej dodatkowej armii, jeśli zasady na to pozwalają (np. poprzez opcję „Dodaj armię” dla sojuszników)[[15]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Najpierw%20nag%C5%82%C3%B3wek%20armii%2C%20potem%20lista,jednostk%C4%99%20do%20armii%E2%80%9D%2C%20%E2%80%9Edodaj%20armi%C4%99%E2%80%9D). Po dodaniu jednostek do listy, poniżej wyświetla się zestawienie wybranych oddziałów: każda pozycja pokazuje nazwę jednostki, wybraną liczbę modeli w oddziale, ich wyposażenie (możliwość zmiany domyślnej broni na inną dostępną), oraz koszt punktowy – rozbity na koszt modeli, koszt broni dodatkowej i ewentualnych zdolności[[16]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Potem%20lista%20dodanych%20jednostek%20z,modelu%2C%20oddzia%C5%82u%2C%20broni%20i%20zdolno%C5%9Bci). Użytkownik może modyfikować liczbę modeli w oddziale, dodawać temu oddziałowi dodatkowe opcje wyposażenia czy zdolności (jeśli są opcjonalne) – system automatycznie przeliczy koszt. Jeśli przekroczony zostanie limit punktów, odpowiednie ostrzeżenie zostanie wyświetlone. Na stronie kreatora będzie również przycisk „Eksportuj listę”, kierujący do podstrony lub akcji generującej PDF/Excel. - **Widok eksportu (PDF/druk):** po wybraniu eksportu listy użytkownik otrzyma wygenerowany raport. Może on być pokazany na stronie (podgląd wydruku) w formie przyjaznej do wydrukowania, z opcją pobrania jako PDF. Alternatywnie, kliknięcie może od razu inicjować pobranie pliku PDF. Format raportu będzie zbliżony do dostarczonego przykładu – dostępne będą dwa style: „ramka” (gdzie każda jednostka wraz z statystykami i wyposażeniem jest ujęta w osobnym polu, przypominającym kartę lub ramkę, co ułatwia odczyt w trakcie gry) oraz „lista” (prostsze zestawienie tabelaryczne całej armii pod rząd, bardziej zwarte)[[5]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Eksport%3A). Oba te formaty będą zoptymalizowane pod wydruk na A4. Dodatkowo możliwe będzie uzyskanie eksportu w układzie zgodnym z Excel – np. tabela podobna do oryginalnego arkusza, dla użytkowników chcących dalej edytować listę poza aplikacją[[5]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Eksport%3A).

Frontend będzie komunikować się z backendem poprzez standardowe zapytania HTTP (formularze dla operacji typu dodaj/edytuj lub żądania GET/POST fetch API jeśli zdecydujemy się na dynamiczne odświeżanie fragmentów strony). Ponieważ interfejs ma być dostępny przez przeglądarkę, istotne będzie zapewnienie automatycznej **skalowalności i responsywności** – użycie gotowego CSS frameworka (np. Bootstrap 5) pozwoli łatwo dostosować widoki do różnych rozdzielczości[[10]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Skalowalno%C5%9B%C4%87%20do%20wy%C5%9Bwietlania%20na%20ekranie,panoramicznym%2C%20laptopie%2C%20tablecie%2C%20telefonie). Dzięki temu aplikacja będzie czytelna i wygodna w obsłudze zarówno na dużym ekranie, jak i na smartfonie.

## 2. Kroki implementacji (etapy)

Realizację projektu warto podzielić na niewielkie etapy, z których każdy można zaimplementować i przetestować osobno – tak, by ewentualnie wygenerować kod w pojedynczych krokach. Poniżej zaproponowano listę kolejnych kroków rozwojowych:

1. **Przygotowanie środowiska i inicjalizacja projektu:** Utwórz repozytorium projektu i wirtualne środowisko Python. Zainstaluj podstawowe zależności (framework webowy, np. Flask, oraz inne potrzebne biblioteki jak SQLAlchemy do bazy danych, biblioteki do generowania PDF/Excel itp.). Zweryfikuj, że aplikacja „Hello World” uruchamia się lokalnie.
2. **Konfiguracja podstawowa aplikacji:** Załóż główny moduł aplikacji (np. plik app.py dla Flask lub projekt Django) oraz plik konfiguracyjny (np. config.py). Skonfiguruj połączenie z bazą SQLite (np. ustaw ścieżkę do pliku .db w konfiguracji) i wygeneruj pustą bazę. Ustaw podstawowe zmienne konfiguracyjne, jak sekret dla sesji (wymagany do logowania), tryb debug itp.
3. **Model bazy danych:** Zdefiniuj modele danych odpowiadające głównym obiektom domeny: Użytkownik (profil), Broń, Zdolność, Jednostka, Armia, ListaArmii (rozpiska). Określ klucze główne/obce i relacje między tymi modelami (np. Jednostka ma klucz obcy do Armii, ListaArmii zawiera wiele jednostek z referencjami do konkretnych wpisów Jednostka, Broń może być przypisana do Jednostki jako domyślna itd.). Ujmij mechanizmy dziedziczenia: np. dodaj opcjonalne pole parent\_id dla Armii/Jednostki/Broni, aby wskazywać obiekt bazowy, z którego dany obiekt pochodzi (to umożliwi odtworzenie hierarchii). Zaimplementuj metodę inicjalizującą bazę (np. tworzącą tabele poprzez ORM) i ewentualnie wstaw kilka przykładowych danych bazowych (np. konto administratora, podstawowe zdolności, przykładowe bronie).
4. **Moduł logowania i autoryzacji:** Dodaj model Użytkownika z polami login i hasło (zahashowane) oraz ewentualnie typem (np. admin lub zwykły). Zaimplementuj mechanizm logowania oraz wylogowania – np. za pomocą Flask-Login lub własnoręcznie poprzez przechowywanie ID użytkownika w sesji. Zapewnij widok logowania (formularz) i rejestracji nowego użytkownika, o ile potrzebne. Na tym etapie można przyjąć, że każdy zalogowany użytkownik ma dostęp do swojej przestrzeni danych, a niezalogowani korzystają z konta wspólnego (np. guest). Nie implementuj skomplikowanej weryfikacji – wystarczy proste sprawdzenie hasła, zgodnie z założeniem zaufanego dostępu[[1]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Mo%C5%BCliwo%C5%9B%C4%87%20zalogowania%20si%C4%99,link%20do%20aplikacji%20jest%20zaufany).
5. **Funkcje zbrojowni (CRUD dla broni):** Zaimplementuj widoki (endpointy) i szablony stron dla listy broni oraz formularzy dodawania/edycji. Zacznij od backendu: stwórz np. blueprint armory (dla Flask) lub odpowiednie widoki (Django) z następującymi funkcjami: lista broni (GET – wyświetla wszystkie bronie z bazy), dodaj nową broń (GET wyświetla formularz, POST zapisuje nowy wpis do bazy), edytuj broń (GET formularz z istniejącymi danymi, POST – zapis zmian), usuń broń (POST/DELETE – usuwa wpis jeśli nie jest używany). Następnie przygotuj szablony HTML: tabela z broniami oraz formularz. Przetestuj dodawanie, edycję, usuwanie broni – upewnij się, że dane trafiają do bazy i są trwałe (sprawdź ponowne uruchomienie aplikacji, dane powinny zostać zachowane[[17]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Baza%20danych%20pozwalaj%C4%85ca%20zachowa%C4%87%20dane,pomi%C4%99dzy%20sesjami)).
6. **Funkcje armii i jednostek:** Podobnie jak wyżej, stwórz moduł/bluprint armies do zarządzania armiami. Zaimplementuj: stronę listy armii (wyświetla istniejące armie, z opcją dodania nowej), formularz tworzenia nowej armii (nazwa armii, wybór wariantu zasad jeżeli dotyczy), edycja armii (zmiana nazwy, ewentualnie zmiana wariantu zasad). Następnie obsłuż jednostki w ramach armii: na stronie edycji armii pokaż listę jednostek danej armii oraz przyciski „Dodaj jednostkę” przy armii i „Edytuj/Usuń” przy każdej jednostce. Zaimplementuj formularz dodawania/edycji jednostki – pola: nazwa jednostki, Jakość, Obrona, Wytrzymałość, lista zdolności (np. wielokrotny wybór lub checkboxy) oraz wybór domyślnej broni. W momencie dodawania broni skorzystaj z danych zbrojowni: np. pozwól wybrać spośród broni globalnych lub wpisz nową nazwę broni, co spowoduje utworzenie nowego wpisu (tu możesz obsłużyć to tak, że jeśli w formularzu wpisano nazwę, której nie ma, to po zapisie tworzysz nową broń z tą nazwą). Upewnij się, że relacje są poprawnie zapisywane (np. nowa jednostka ma przypisane id armii; jeśli tworzona jest jednostka jako wariant innej, ustawiasz jej parent\_id na oryginalną jednostkę).
7. **Mechanizm dziedziczenia i zależności:** Rozbuduj logikę modułów zbrojowni/armii tak, aby uwzględniały hierarchiczność. Przykładowo, jeżeli użytkownik edytuje broń **globalną** na poziomie armii (czyli chce zmodyfikować parametry broni tylko dla swojej armii), w kodzie utwórz nowy obiekt broni, ustaw jego parent\_id na broń globalną i powiąż z daną armią jako jej lokalna broń. W ten sposób zmiany nie wpływają na broń globalną, a jedynie na wariant w konkretnej armii[[18]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Hierarchiczno%C5%9B%C4%87%2Fdziedziczenie%3A%20bronie%2C%20jednostki%20i%20armie,je%20do%20rodzica%20usuwanego%20elementu). Podobnie dla jednostek: jeśli armia B jest wariantem armii A, to jednostki armii B mają pole parent\_id wskazujące na jednostki z A. Zaimplementuj funkcję do usuwania elementów hierarchii: np. usunięcie armii powinno automatycznie usuwać wszystkie jednostki, które do niej należą, o ile nie są wykorzystywane w żadnej liście (rozpisce)[[19]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=znale%C5%BA%C4%87%20wszystkie%20dzieci%20i%20przekierowa%C4%87,je%20do%20rodzica%20usuwanego%20elementu). Analogicznie usunięcie broni powinno sprawdzić, czy nie jest referencjonowana; jeśli nie – można ją bezpiecznie skasować. Nieużywane obiekty podrzędne przekieruj lub usuń zgodnie z zasadami (można też oznaczać pewne rekordy jako usunięte/archiwalne, by nie łamać referencji, ale w prostym rozwiązaniu usuwanie cascade może być akceptowalne).
8. **Kreator listy armii (rozpiski):** Zaimplementuj moduł rosters lub rozszerz moduł armii o funkcje tworzenia list. Stwórz widok „Utwórz nową listę” – najpierw wybór armii bazowej (lub kilku, jeśli sojusze). Następnie w sesji lub stanie aplikacji twórz nowy obiekt ListaArmii i pozwól użytkownikowi dodawać do niej jednostki. Można to zrobić poprzez dynamiczny formularz: np. lista dostępnych jednostek z przyciskiem „Dodaj” obok każdej – po kliknięciu pojawia się w sekcji "Twoja rozpiska" nowy wpis jednostki z polami edycji (liczebność, wyposażenie). Zaimplementuj na backendzie endpoint do dodawania jednostki do listy (POST z identyfikatorem jednostki i listy), zwiększania/zmniejszania liczebności, usuwania jednostki z listy. Równocześnie obliczaj sumę punktów: przy każdym dodatniu/zmianie jednostki wywołaj funkcję liczącą koszt (na podstawie statystyk jednostki, jej broni i zdolności) pomnożony przez liczbę modeli. Wynik sumuj do całkowitego kosztu listy i porównaj z limitem (jeśli ustawiono). Na frontendzie wyświetlaj tę sumę i np. oznacz na czerwono, gdy przekracza limit.
9. **Walidacja zasad i kosztów:** Po zbudowaniu podstawowego kreatora, dodaj sprawdzanie reguł szczegółowych zgodnie z *Moje OPR*. Przykładowo, uniemożliw dodanie więcej niż jednej zdolności aktywnej na jednostkę czy łączenie aury + zdolności aktywnej na jednym modelu[[20]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Znaczniki%20zdolno%C5%9Bci%3A%20Ka%C5%BCdy%20odzia%C5%82%20mo%C5%BCe,albo%20zdolno%C5%9B%C4%87%20aktywn%C4%85%2C%20albo%20aur%C4%99). Wbuduj w kalkulator kosztów formuły z dokumentu zasad: koszt bazowy (wspomniany wyżej 5 \* modJakość \* modObrona \* modWytrzymałość), koszt broni (np. zależny od zasięgu, liczby ataków, AP – szczegółowe formuły należy zaimplementować zgodnie z dokumentacją, ewentualnie przenosząc logikę z arkusza kalkulacyjnego i jego makr), koszt zdolności (np. każdy punkt Wytrzymałości dodaje koszt każdej zdolności pasywnej[[7]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Koszt%20zdolno%C5%9Bci%20pasywnych%20nale%C5%BCy%20zap%C5%82aci%C4%87,za%20ka%C5%BCdy%20punkt%20wytrzyma%C5%82o%C5%9Bci), a zdolności z parametrem X mają swoje przeliczniki[[9]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Transport,Lataj%C4%85cy%2C%201%2C25%20Szybki%2FZwinny%2C%201%20pozosta%C5%82e)). Przetestuj kilka scenariuszy, porównując wyniki z arkuszem Excel *Moja\_OPR\_list.xlsm*, aby upewnić się, że kalkulacje się zgadzają i że aplikacja poprawnie obsługuje wszystkie przypadki (np. jednostka bez broni dystansowej jest traktowana jako *wojownik* itp.[[21]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=modyfikatory%20zdolno%C5%9Bci)).
10. **Funkcja eksportu PDF:** Skonfiguruj bibliotekę do generowania PDF. Możliwe podejścia: wygenerowanie PDF z kodu HTML/CSS (np. przy użyciu bibliotek typu WeasyPrint, xhtml2pdf albo wkhtmltopdf) lub bezpośrednie utworzenie dokumentu PDF (np. ReportLab) na podstawie danych. Najprostszym rozwiązaniem może być przygotowanie szablonu HTML dla listy armii w trybie do druku (CSS ustawiony pod format A4, układ kolumnowy lub ramki) i użycie narzędzia, które konwertuje ten HTML na PDF. Zaimplementuj endpoint /export/pdf/<lista\_id> który zwróci plik PDF do pobrania. Upewnij się, że zawartość PDF zawiera wszystkie potrzebne informacje: nazwa armii, każda jednostka z pełnymi statystykami, wypunktowane zdolności, uzbrojenie oraz koszty częściowe i sumaryczne. Przetestuj otwierając wygenerowany PDF w przeglądarce lub Acrobat Reader.
11. **Funkcja eksportu Excel (opcjonalnie):** Jeśli wymagana, zaimplementuj generowanie pliku .xlsx dla listy. Można użyć biblioteki OpenPyXL lub Pandas do stworzenia arkusza. W pliku Excel umieść podobne dane jak na wydruku – np. jeden arkusz z listą jednostek (wiersz na jednostkę z kolumnami: nazwa, liczba modeli, statystyki, broń, zdolności, koszt) oraz ewentualnie dodatkowy arkusz „Zbrojownia” z listą użytych profili broni. Ponieważ użytkownik wspomniał, że eksport do Excela jest opcjonalny, ten krok można zrealizować na końcu, po dopięciu głównych funkcji. Endpoint może być analogiczny do PDF: /export/xlsx/<lista\_id>, zwracający plik Excel do pobrania.
12. **Interfejs użytkownika i szlify graficzne:** Gdy backend jest gotowy, dopracuj frontend. Dodaj style CSS i skrypty JavaScript (jeśli potrzebne) dla lepszej użyteczności. Na tym etapie warto włączyć framework CSS (np. dołączyć Bootstrap przez CDN) i przejrzeć wszystkie strony na różnych urządzeniach/rozdzielczościach, dostosowując układ elementów aby *layout* był czytelny (np. na małym ekranie tabela broni może przewijać się na boki, menu może zmienić się w tzw. „hamburger menu” itp.). Zaimplementuj również komunikaty błędów/sukcesów dla akcji (np. „dodano jednostkę”, „brak nazwy – wprowadź nazwę broni” itp.), walidację formularzy po stronie klienta (HTML5/JS) i serwera. Dodaj linki/nawigację między stronami dla wygody (np. powrót do listy, przycisk na górze do strony głównej, wylogowanie itp.).
13. **Testy i weryfikacja zgodności z zasadami:** Przeprowadź kompleksowe testy aplikacji. Sprawdź, czy możliwe jest odtworzenie wszystkich scenariuszy tworzenia armii i list zgodnie z dokumentacją. Przetestuj mechanizm dziedziczenia (czy edycja broni lokalnej nie zmienia globalnej, usunięcie armii usuwa jej jednostki, itp.). Upewnij się, że dla wielu użytkowników dane są odseparowane (użytkownik A nie widzi prywatnych armii użytkownika B). Przejrzyj wygenerowane PDF/Excel i porównaj z oczekiwaniami (np. z przykładowym wydrukiem dostarczonym przez użytkownika) – popraw ewentualne rozbieżności w formatowaniu lub zawartości.
14. **Przygotowanie do wdrożenia:** Zaktualizuj dokumentację projektu (README.md) o instrukcje instalacji i uruchomienia. Zamroź zależności do pliku requirements.txt. Rozważ dodanie prostego mechanizmu inicjalizacji bazy przy starcie (np. jeśli brak pliku bazy, to utwórz tabele i załaduj dane początkowe: konto admin, podstawowe zdolności, przykładowe armie itp., co ułatwi pierwsze uruchomienie).
15. **Wdrożenie produkcyjne:** Przenieś aplikację na docelowy serwer i uruchom ją w trybie produkcyjnym (szczegóły w kolejnym punkcie). Po wdrożeniu przetestuj raz jeszcze kluczowe funkcje już na serwerze (przez internet), aby upewnić się, że wszystko działa w przewidzianym środowisku.

Każdy z tych etapów jest na tyle ograniczony, że można go zrealizować (lub wygenerować) niezależnie w ramach jednego polecenia czy sesji, co ułatwi stopniowe budowanie aplikacji oraz kontrolę wersji i testowanie.

## 3. Struktura projektu i pliki konfiguracyjne

Poniżej przedstawiono proponowaną strukturę katalogów i plików projektu (dla przykładu przyjmijmy Flask, struktura dla Django będzie nieco inna z podziałem na aplikacje, ale idee konfiguracyjne podobne):

projekt\_opr/  
├── app.py # Główny plik uruchamiający aplikację (inicjalizacja Flask, registracja blueprintów)  
├── config.py # Plik konfiguracji aplikacji (ustawienia bazy danych, sekret, opcje konfiguracyjne)  
├── requirements.txt # Lista zależności Pythona (framework webowy, biblioteki PDF/Excel, ORM itp.)  
├── /data/  
│ └── opr.db # Plik bazy danych SQLite (przechowuje wszystkie dane aplikacji)  
├── /migrations/ # (opcjonalnie) skrypty migracji bazy danych, jeśli używany jest mechanizm migracji (np. Flask-Migrate)  
├── /opr/ # Pakiet z kodem aplikacji (można nazwać inaczej, np. core/)  
│ ├── \_\_init\_\_.py # Inicjalizacja pakietu (inicjalizacja obiektów Flask, DB itp.)  
│ ├── models.py # Definicje modeli ORM (Użytkownik, Armia, Jednostka, Broń, Zdolność, ListaArmii, itp.)  
│ ├── forms.py # Definicje formularzy (jeśli używamy Flask-WTF do walidacji) lub schematów (dla Django – w forms.py lub w widokach)  
│ ├── utils.py # Funkcje pomocnicze (np. obliczanie kosztów na podstawie statystyk, formatowanie tekstu do eksportu)  
│ ├── auth.py # Widoki/endpointy związane z uwierzytelnianiem (logowanie, wylogowanie, rejestracja)  
│ ├── armory.py # Widoki/endpointy zbrojowni (lista broni, dodaj/edytuj/usuń broń)  
│ ├── armies.py # Widoki/endpointy armii i jednostek (lista armii, edycja armii, dodaj/edytuj jednostkę)  
│ ├── rosters.py # Widoki/endpointy list armijnych (tworzenie i edycja rozpisek, eksport)  
│ └── templates/ # Szablony HTML  
│ ├── base.html # Główny szkielet strony (nagłówek, menu nawigacyjne, blok na treść)  
│ ├── index.html # Strona główna (dashboard z listami armii i rozpisek)  
│ ├── login.html # Formularz logowania  
│ ├── armory\_list.html # Strona listy broni (tabela)  
│ ├── armory\_form.html # Formularz dodawania/edycji broni  
│ ├── armies\_list.html # Strona listy armii  
│ ├── army\_edit.html # Widok edycji konkretnej armii (z listą jednostek)  
│ ├── unit\_form.html # Formularz dodawania/edycji jednostki  
│ ├── roster\_edit.html # Widok edycji listy armii (wybór jednostek, ustawienia)  
│ ├── roster\_print.html # Szablon podglądu wydruku listy (ładny układ do PDF/drukarki)  
│ └── ... (ew. inne, np. szablon modalnego okna do wyboru broni, itp.)  
├── /static/ # Pliki statyczne (CSS, JS, obrazy)  
│ ├── css/  
│ │ └── style.css # Dodatkowe style (nadpisujące/domykające framework CSS)  
│ ├── js/  
│ │ └── scripts.js # Skrypty frontendu (opcjonalnie, np. do dynamicznej obsługi formularzy)  
│ └── images/  
│ └── ... # Ewentualne obrazy (np. logo gry, tła, ikony)  
└── /docs/ # Dokumentacja, pliki pomocnicze (np. załączone PDFy z zasadami, przykładowe listy itp.)  
 └── Moje\_OPR\_zasady.pdf # (opcjonalnie) Kopia zasad w PDF dla odniesienia

W powyższej strukturze wyróżniono kluczowe elementy: - **Pliki konfiguracyjne:** config.py zawiera ustawienia aplikacji. Typowe opcje to DATABASE\_URI (ścieżka do pliku SQLite, np. sqlite:///data/opr.db), SECRET\_KEY (losowy klucz używany do podpisywania ciasteczek sesji – zapewnia bezpieczeństwo sesji), opcje debugowania (np. DEBUG=True w trakcie developmentu) i ewentualnie ustawienia związane z eksportem (np. ścieżka do narzędzia wkhtmltopdf, jeśli jest używane). W przypadku Django, odpowiednikiem będzie plik settings.py z analogicznymi ustawieniami. - **Plik główny uruchamiający (entry point):** app.py tworzy instancję aplikacji Flask, ładuje konfigurację z config.py, inicjalizuje np. obiekt bazy danych (SQLAlchemy) i rejestruje blueprinty (auth, armory, armies, rosters). W przypadku Django takim plikiem jest manage.py + konfiguracja w settings.py, a aplikacje byłyby zorganizowane w katalogach pod /opr/. - **Modele i logika domenowa:** w models.py trzymamy klasy odzwierciedlające tabele bazy danych. Każda klasa ma atrybuty (kolumny) i może mieć metody pomocnicze, np. metoda compute\_cost() dla jednostki, która sumuje koszt bazowy, broni i zdolności[[6]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Koszt%20modelu%20jest%20sum%C4%85%20kosztu,kosztu%20broni%20i%20kosztu%20zdolno%C5%9Bci). Możliwe, że modele zostaną podzielone na kilka plików (np. models/army.py, models/weapon.py itd.) dla czytelności w większym projekcie – to zależy od preferencji. - **Szablony HTML:** w folderze templates/ znajdują się pliki .html definiujące wygląd poszczególnych stron. Używamy mechanizmu dziedziczenia szablonów – base.html zawiera ogólny układ (np. nagłówek z tytułem aplikacji, menu użytkownika, kontener na treść), a pozostałe szablony rozszerzają go wypełniając swoje sekcje treści. Np. armory\_list.html rozszerza base.html i w sekcji treści wstawia tabelę z listą broni. - **Pliki statyczne:** w static/ umieszczone są elementy niezmienne wysyłane do przeglądarki wprost. Będziemy tu mieć plik CSS (można załączyć Bootstrap z CDN, ale własny style.css może zawierać drobne poprawki np. dla wydruku czy dodatkowe stylowanie przycisków) oraz plik JS z kodem frontendu (jeśli potrzebny – np. skrypty do dynamicznego dodawania pól w formularzach, potwierdzenia usunięcia, itp.). Obrazy, jeśli aplikacja używa (np. logo lub ikony), też trafią tutaj. - **Baza danych:** katalog data/ przechowuje plik SQLite (opr.db). Dobrze, by aplikacja zakładała go automatycznie, jeśli nie istnieje (przy starcie sprawdzać, czy plik jest obecny, ewentualnie utworzyć tabele). W trakcie developmentu używana będzie ta baza plikowa; w przyszłości, dla większej liczby użytkowników, można rozważyć migrację na serwer baz danych (PostgreSQL, MySQL), ale dla potrzeb prywatnego serwera SQLite jest wystarczający i prosty w utrzymaniu.

**Pliki konfiguracyjne dodatkowe:** Poza config.py i requirements.txt, przy wdrożeniu produkcyjnym mogą pojawić się: - wsgi.py (dla deploymentu na serwerach kompatybilnych z WSGI, np. użylibyśmy Gunicorn, który będzie odwoływał się do aplikacji poprzez ten plik). - .env (plik środowiskowy z hasłami/kluczami – nieobowiązkowe, ale wygodne do oddzielenia np. sekretnego klucza czy danych dostępowych od kodu; można użyć python-dotenv by go wczytać). - Procfile (jeśli wdrażamy na Heroku lub podobnych PaaS). - pliki konfiguracyjne serwera WWW (np. konfiguracja Nginx, jeśli będzie reverse proxy) – to jednak poza samym projektem aplikacji.

## 4. Środowisko uruchomieniowe

Aplikacja zostanie napisana w **Python 3** (należy użyć aktualnej wersji Python 3.10+). Wymagane będą następujące komponenty i biblioteki środowiskowe:

* **Framework webowy:** Flask (wersja 2.x) lub alternatywnie Django (4.x). Flask jest lekki i prosty w użyciu, co pasuje do projektu prowadzonego przez indywidualnego użytkownika; Django zapewnia więcej gotowych rozwiązań (np. panel admin, system uwierzytelniania), ale jest bardziej rozbudowany – wybór zależy od preferencji, jednak w planie zakładamy Flask dla przejrzystości.
* **Baza danych:** SQLite (silnik wbudowany w Pythona, brak dodatkowych instalacji). Biblioteka do obsługi ORM – SQLAlchemy (dla Flask, zwykle przez flask-sqlalchemy) lub wbudowany ORM Django. Alternatywnie można używać czystego SQLite (biblioteka sqlite3) jeśli ORM wydaje się zbyt skomplikowany, jednak ORM ułatwi implementację dziedziczenia i relacji.
* **Biblioteki do PDF:** np. WeasyPrint (do konwersji HTML/CSS na PDF) lub xhtml2pdf. WeasyPrint wymaga zainstalowania bibliotek systemowych (Cairo, Pango itp.), a wkhtmltopdf wymaga instalacji binarki wkhtmltopdf. Można też użyć ReportLab (czysto pythonowa biblioteka do rysowania PDF), lecz generowanie layoutu „ramki” od zera w PDF byłoby bardziej pracochłonne niż konwersja gotowego HTML. Zakładamy użycie podejścia HTML -> PDF dla wygody projektowania szablonu.
* **Biblioteki do Excela:** opcjonalnie OpenPyXL lub XlsxWriter – obie pozwalają na tworzenie plików .xlsx. Można też użyć Pandas DataFrame do łatwego eksportu, ale zważywszy na specyfikę formatu (trzeba odwzorować strukturę listy armii), bezpośrednie użycie OpenPyXL może być najwygodniejsze. Plik Excel nie wymaga żadnych zewnętrznych programów – generowany jest wprost przez Pythona.
* **Inne biblioteki Python:** Flask-Login (jeśli Flask, do obsługi logowania użytkowników), Flask-WTF (formularze i walidacja), wtforms (dla Flask-WTF), dataclasses lub pydantic (opcjonalnie, do walidacji danych wejściowych), python-dotenv (jeśli korzystamy z pliku .env do konfiguracji), PyYAML/JSON (jeśli konfiguracja zasad gry będzie kiedyś w osobnym pliku). W razie korzystania z Django: django-allauth (opcjonalnie, do rejestracji), ale w uproszczeniu standardowy moduł auth Django wystarczy.
* **Frontend i środowisko przeglądarki:** Aplikacja będzie działać w przeglądarce – zalecane aktualne wersje Chrome/Firefox/Edge na PC i mobilne. Do poprawnego działania frontendu należy uwzględnić dołączenie plików JS/CSS: w szczególności Bootstrap CSS (np. z CDN) oraz ewentualnie Bootstrap JS (jeśli korzystamy z komponentów jak dropdown menu, modale). Alternatywnie można użyć frameworka CSS innego lub nawet własnego, ale gotowe rozwiązanie przyspieszy uzyskanie responsywności.
* **System operacyjny:** Dowolny, który obsłuży powyższe (Linux, Windows, macOS). Dla środowiska produkcyjnego najczęściej wybrany będzie Linux (np. Ubuntu 22.04 LTS) – jest stabilny pod serwery Pythonowe. Na serwerze trzeba zainstalować interpreter Python oraz potrzebne pakiety (najlepiej w wirtualnym środowisku). Jeśli PDF generowany jest przez wkhtmltopdf, należy zainstalować także ten program (np. przez apt). WeasyPrint wymaga z kolei bibliotek systemowych (Cairo, libpango, libgdk-pixbuf2 itp., instalowanych przez menedżer pakietów).
* **Wymagania sprzętowe:** Niewielkie – aplikacja nie jest zasobożerna. Wystarczy nawet 1 CPU i 512 MB RAM (np. Raspberry Pi czy mały VPS poradzi sobie, choć generowanie PDF może chwilę potrwać). Baza SQLite utrzymuje dane w pliku, co przy jednoczesnym dostępie wielu użytkowników może powodować krótkie blokady zapisu – ale ponieważ użytkowników jest mało (prywatna aplikacja), nie powinno to stanowić problemu.

Podsumowując, potrzebne jest środowisko z Pythonem i kilkoma bibliotekami. Wszystkie zależności Pythonowe zostaną wyszczególnione w requirements.txt, co ułatwi instalację (pip install -r requirements.txt). Ważne jest również posiadanie na serwerze uprawnień do zapisu w katalogu z bazą danych oraz generowania plików (PDF/Excel) – np. aplikacja będzie zapisywać pliki tymczasowo przed wysyłką do użytkownika.

## 5. Wskazówki dotyczące wdrożenia na serwerze

Wdrożenie aplikacji na prywatnym serwerze wymaga kilku kroków konfiguracyjnych. Zakładamy, że użytkownik dysponuje serwerem (lub komputerem) stale podłączonym do Internetu, z zainstalowanym Pythonem. Oto zalecane podejście do uruchomienia aplikacji w trybie produkcyjnym:

* **Klonowanie/Instalacja kodu:** Umieść kod źródłowy aplikacji na serwerze. Można to zrobić poprzez klonowanie z repozytorium git lub skopiowanie plików przez SCP/FTP. Upewnij się, że struktura plików jest spójna z założeniami (zwłaszcza plik bazy danych – jeśli przenosisz z lokalnego komputera jakieś wstępne dane, skopiuj również plik .db).
* **Wirtualne środowisko:** Na serwerze utwórz wirtualne środowisko Pythona (poleceniem python3 -m venv venv) i aktywuj je. Następnie zainstaluj zależności: pip install -r requirements.txt. Jeśli jakieś biblioteki mają zależności systemowe (np. WeasyPrint, wkhtmltopdf), zainstaluj je poprzez systemowy menedżer pakietów (np. apt-get dla Debiana/Ubuntu) – szczegółowe instrukcje zazwyczaj są w dokumentacji tych narzędzi.
* **Konfiguracja zmiennych środowiskowych:** Ustaw zmienne potrzebne aplikacji. Dla Flask można ustawić FLASK\_APP=app.py oraz FLASK\_ENV=production (lub po prostu w kodzie app.run(debug=False)). Upewnij się, że SECRET\_KEY jest ustawiony na unikalną wartość (może być w pliku config.py lub w zmiennej środowiskowej i wczytywany przez aplikację). Jeśli korzystasz z .env, umieść ten plik na serwerze.
* **Uruchomienie serwera aplikacji:** Najprostszym sposobem startu jest użycie wbudowanego serwera deweloperskiego Flask (flask run --host=0.0.0.0 --port=5000). Jednak dla stabilności i wielowątkowości zaleca się użycie serwera WSGI. Przykładowo, zainstaluj Gunicorn (pip install gunicorn) i uruchom aplikację komendą: gunicorn -w 4 -b 0.0.0.0:5000 app:app. To wystartuje 4-workerowy serwer obsługujący aplikację (nazewnictwo app:app zakłada, że w pliku app.py jest zmienna app = instancja Flask). Jeśli używasz Django, analogicznie użyj gunicorn wskazując obiekt WSGI: gunicorn projekt\_opr.wsgi:application.
* **Reverse proxy (opcjonalnie):** Jeśli aplikacja ma działać na standardowym porcie HTTP (80) i ewentualnie obsługiwać SSL (HTTPS), warto postawić przed Gunicornem serwer reverse-proxy, np. Nginx lub Apache. Nginx może przekazywać zapytania z portu 80 do aplikacji działającej na porcie 5000. Konfiguracja Nginx powinna uwzględnić też serwowanie plików statycznych (można je serwować bezpośrednio przez Nginx z katalogu /static, co odciąży aplikację). W przypadku braku doświadczenia z Nginx, a jeśli aplikacja będzie używana tylko sporadycznie przez zaufanych użytkowników, można rozważyć pozostanie przy samym Gunicornie na niestandardowym porcie lub używanie tunelu SSH/VPN zamiast wystawiać aplikację na świat – w zależności od wymaganego poziomu dostępu.
* **Dostęp przez internet:** Zakładając, że serwer ma publiczne IP lub domenę – skonfiguruj DNS lub używaj bezpośrednio adresu IP:port. Upewnij się, że zapora sieciowa (firewall) pozwala na ruch do wybranego portu. Jeśli korzystasz z portu 80/443 za Nginx, otwórz te porty. Dla testów możesz użyć narzędzi typu ngrok (tunelowanie) lub lokalnego port forwarding, ale docelowo stały dostęp wymaga konfiguracji sieciowej na serwerze.
* **Wdrożenie bazy danych:** Jeśli na etapie developmentu używano wypełnionej bazy SQLite, zdecyduj czy przenieść ją na serwer (zachowując dotychczasowe dane) czy zacząć od pustej bazy. W prywatnej aplikacji pewnie chcesz zachować to, co już zdefiniowałeś (np. listy armii, bronie). Skopiuj plik .db na serwer do wskazanej ścieżki. Upewnij się, że użytkownik procesu serwera ma prawo zapisu do pliku/katalogu bazy (SQLite zapisuje do pliku, więc np. user www-data musi mieć prawo zapisu w /data/opr.db).
* **Uruchomienie jako usługa:** Aby aplikacja działała stale, skonfiguruj ją jako usługę systemową. W systemd możesz stworzyć plik jednostki (service) odpalający Gunicorn z odpowiednimi parametrami przy starcie systemu. Dzięki temu po ewentualnym restarcie serwera aplikacja sama się uruchomi. W pliku service ustaw WorkingDirectory na katalog projektu, user na odpowiedniego użytkownika (nie root, najlepiej dedykowany user aplikacji), oraz ExecStart z komendą gunicorn jak wyżej. Następnie włącz usługę (systemctl enable app.service) i uruchom (systemctl start app.service).
* **Logi i monitoring:** Sprawdź logi aplikacji (konsoli lub konfigurując Gunicorn by logował do pliku) pod kątem błędów po wdrożeniu. Upewnij się, że generowanie PDF działa na serwerze – czasem może brakować fontów lub uprawnień do odpalania narzędzia do PDF (szczególnie wkhtmltopdf wymaga dostępu do X11 lub trybu headless – można mu to zapewnić parametrami). Przetestuj równoczesne korzystanie (dwóch użytkowników naraz dodających różne rzeczy) – obserwuj czy brak konfliktów (SQLite powinien sekwencyjnie obsłużyć zapisy, a odczyty równolegle).
* **Kopia zapasowa:** Na koniec, skonfiguruj mechanizm robienia kopii bazy danych (choćby manualnie co jakiś czas kopiując plik .db, lub napisanie małego skryptu backupującego codziennie dane). Ponieważ to prywatny serwer, uproszczone podejście jest akceptowalne – warto jednak zabezpieczyć się przed utratą danych po dłuższym czasie użytkowania.

Wdrożenie w takiej formie zapewni, że aplikacja będzie dostępna pod ustalonym adresem (np. http://twoj-serwer:5000 lub poprzez domenę) dla wszystkich uprawnionych użytkowników. Ponieważ zakładamy zaufany krąg odbiorców, nie wdrażamy skomplikowanych zabezpieczeń, ale podstawowe środki (jak unikalne hasła dla kont, niestosowanie domyślnego sekretu Flask) powinny być zachowane. W razie potrzeby szerszego udostępnienia aplikacji, można wprowadzić dodatkowe zabezpieczenia (np. podstawowa autoryzacja HTTP dla całej aplikacji lub mechanizmy ograniczania rejestracji nowych użytkowników).

## 6. Kluczowe funkcje i zgodność z zasadami gry

Poniżej zebrano szczególne wymagania funkcjonalne, o których wspomniano, wraz z informacją jak zostaną zrealizowane w projekcie:

* **Hierarchiczność obiektów i dziedziczenie:** Aplikacja będzie wspierać hierarchiczne zależności między obiektami, tak aby bronie, jednostki i armie mogły istnieć w wersjach bazowych i pochodnych. W praktyce oznacza to, że np. armia może być wariantem innej armii – w bazie nowa armia otrzyma referencję parent\_id do armii bazowej. Wszystkie jednostki utworzone w wariancie początkowo dziedziczą statystyki i wyposażenie z jednostek bazowych. Jeśli użytkownik edytuje element pochodny (np. modyfikuje statystyki jednostki w armii wariantu lub zmienia parametry broni, która została odziedziczona z globalnej zbrojowni), system nie nadpisuje obiektu bazowego, tylko tworzy nowy wpis w bazie, powiązany z oryginałem jako jego dziecko[[18]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Hierarchiczno%C5%9B%C4%87%2Fdziedziczenie%3A%20bronie%2C%20jednostki%20i%20armie,je%20do%20rodzica%20usuwanego%20elementu). W ten sposób zapisywane są tylko zmienione pola, a pola niezmienione można w interfejsie pokazywać z obiektu bazowego (domyślnego). Usuwanie elementów również uwzględnia hierarchię – jeśli usuwamy np. całą armię, aplikacja powinna automatycznie usunąć jednostki należące tylko do niej (dzieci), o ile nie są używane gdzie indziej[[19]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=znale%C5%BA%C4%87%20wszystkie%20dzieci%20i%20przekierowa%C4%87,je%20do%20rodzica%20usuwanego%20elementu). Analogicznie usunięcie globalnej broni powinno przenieść ewentualne warianty w armiach na wyższy poziom lub również je usunąć, jeśli są osierocone. Takie podejście zapewni spójność danych i wygodę – użytkownik może tworzyć warianty bez ręcznego przepisywania wszystkiego, a zmiany w jednym miejscu nie rozwalą całego systemu danych.
* **Edytory armii i broni:** Przewidziane są przyjazne w użyciu edytory do zarządzania zarówno listą armii, jak i listą broni. Ekran **Zbrojownia** (bronie) wyświetli wszystkie dostępne bronie w formie tabeli, pozwalając na szybkie dodanie nowej pozycji lub edycję istniejącej[[12]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Lista%20broni%20%E2%80%93%20uk%C5%82ad%20podobny,jak%20we%20wzorcowym%20Excelu). Użytkownik w formularzu broni poda wszystkie właściwości (nazwę, statystyki) zgodnie ze strukturą danych. Dla ułatwienia, pola mogą być w formularzu opisane tak jak kolumny w arkuszu (np. „Zasięg (cale)”, „Ataki (ilość rzutów)”, „AP (modyfikator przebicia)”, „Specjalne cechy” itp.), co czyni nawiązanie do Excela bezpośrednie[[13]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Zbrojownia%3A). Ekran **Armie** wyświetla listę utworzonych armii z możliwością dodania nowej. W edycji armii można dodać jednostkę – pojawi się formularz bardzo podobny do tego z Excela: nazwa jednostki, statystyki (Jakość, Obrona, Wytrzymałość), a także lista zdolności (do wyboru z dostępnych) i broń domyślna[[2]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Armie%3A). Dla wygody użytkownika podczas wyboru broni dla jednostki, aplikacja zaoferuje listę istniejących broni: najpierw broni domyślnej odziedziczonej (jeśli jednostka jest kopiowana), potem broni przypisanych do tej armii, a następnie całej reszty zbrojowni globalnej[[14]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Lista%20modeli%20w%20armii%3A%20nazwa%2C,podstawowe%20atrybuty%2C%20zdolno%C5%9Bci%2C%20domy%C5%9Blna%20bro%C5%84). Oprócz tego będzie opcja „Dodaj nową broń” bezpośrednio z poziomu edytora jednostki – co umożliwi szybkie wprowadzenie nowego wyposażenia w trakcie definiowania jednostki. Podobnie dla zdolności – jeśli jakaś zdolność nie figuruje na liście (np. w przyszłości dodamy własną), administrator będzie mógł ją dodać do bazy zdolności. W ten sposób edytory zapewnią pełną kontrolę nad zawartością armii i zbrojowni, z zachowaniem logiki dziedziczenia (edycja elementu pochodnego tworzy kopię – użytkownik otrzyma np. pytanie czy „Zmodyfikować tylko w tej armii czy globalnie?” aby świadomie wybrać, co chce zrobić).
* **Zarządzanie profilami użytkowników:** System obsłuży wielu użytkowników jednocześnie, przy czym nie planujemy rozbudowanych ról poza kontem administratora. Każdy użytkownik będzie miał własne konto chronione hasłem (choć można zrezygnować z rejestracji i dostarczać konta ręcznie, skoro dostęp ma być ograniczony). Zgodnie z wymaganiami, niezalogowany użytkownik korzysta z profilu wspólnego (np. gość), natomiast po zalogowaniu użytkownik widzi tylko obiekty utworzone przez siebie lub globalne obiekty wspólne[[1]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Mo%C5%BCliwo%C5%9B%C4%87%20zalogowania%20si%C4%99,link%20do%20aplikacji%20jest%20zaufany). Oznacza to, że dane takie jak armie, bronie, listy będą oznaczone w bazie polem owner (wskazującym ID użytkownika tworzącego). Obiekty stworzone przez administratora mogą być oznaczone jako globalne (wspólne dla wszystkich). Przykładowy scenariusz: Admin dodaje armię „Kosmiczni Marines” wraz z jednostkami – widzą ją wszyscy i mogą na jej podstawie tworzyć rozpiski. Użytkownik Jan tworzy nową armię „Moja Unikalna Armia” – on ją widzi i edytuje, ale inni użytkownicy nie zobaczą jej (chyba że np. admin przeniesie ją do globalnych). Taki model ochroni prywatne prace użytkowników, a zarazem umożliwi współdzielenie zasobów podstawowych. Implementacja opierać się będzie o mechanizmy frameworka – np. dekoratory @login\_required do zabezpieczenia edycji, sprawdzanie uprawnień (czy current\_user jest właścicielem danego obiektu) przed operacją modyfikacji/usunięcia itp. Z racji przyjęcia zaufanego środowiska nie przewiduje się zaawansowanych zabezpieczeń czy ograniczeń – raczej chodzi o wygodę filtrowania widoku dla użytkownika niż o odporność na ataki. Mimo to podstawowe środki (hashowanie haseł, unikanie ujawniania cudzych ID w URL) zostaną zachowane.
* **Przechowywanie danych:** Wszystkie dane aplikacji (armie, jednostki, bronie, profile użytkowników, listy) będą trwale przechowywane w relacyjnej bazie danych SQLite (plik .db). Dzięki temu po ponownym uruchomieniu serwera czy odświeżeniu strony dane nie znikną[[17]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Baza%20danych%20pozwalaj%C4%85ca%20zachowa%C4%87%20dane,pomi%C4%99dzy%20sesjami). SQLite jest łatwe w utrzymaniu – do wykonania kopii zapasowej wystarczy skopiować plik, a do wglądu w dane można użyć dowolnego klienta SQLite. Struktura bazy została zaprojektowana tak, by odwzorować zależności hierarchiczne (self-referencyjne klucze obce) oraz relacje wiele-do-wielu (np. jednostka może mieć wiele zdolności – tu zastosujemy albo tabelę pomocniczą jednostka\_zdolność, albo model Zdolność z parametrem wskazującym typ obiektu i kluczem uniwersalnym – w zależności od preferencji). Ważne jest zapewnienie integralności – np. przy usuwaniu rekordu kaskadowe usunięcie powiązanych lub zablokowanie usunięcia jeśli są zależności. ORM (jeśli użyty) uprości te zadania poprzez odpowiednie ustawienia relacji (np. ondelete dla kluczy obcych jako CASCADE lub poprzez ręczne funkcje kasujące). Dane w bazie będą mogły być również migrowane – użyjemy narzędzia migracyjnego (Flask-Migrate/alembic przy SQLAlchemy lub wbudowanych migracji Django) co umożliwi późniejsze zmiany schematu bez utraty już wprowadzonych informacji.
* **Eksport do PDF/Excel:** Jak wspomniano, kluczową funkcją jest możliwość eksportowania listy armii do PDF (w celu wydruku lub wygodnego przeglądania na ekranie) oraz do pliku Excel (np. dla dalszej edycji lub archiwizacji)[[5]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Eksport%3A). W aplikacji zrealizuje to osobny komponent. Eksport PDF będzie dostępny poprzez przycisk „Eksportuj do PDF” na stronie rozpiski – po kliknięciu aplikacja wygeneruje dokument i zaoferuje pobranie. W generowanym PDF znajdzie się sformatowana lista armii: każda jednostka w osobnej ramce lub wierszu (w zależności od wybranego stylu). Na PDF pojawią się wszystkie istotne dane: statystyki, broń, zdolności oraz rozbicie kosztów i suma punktów. Dzięki odpowiednio przygotowanemu szablonowi (uwzględniającemu podział na kolumny, pogrubienia, ikony dla różnych rodzajów broni być może) wydruk będzie czytelny i estetyczny – co jest ważne dla graczy. Eksport do Excela będzie dostępny opcjonalnie – np. obok PDF przycisk „Eksportuj do .xlsx”. Wygenerowany plik excelowski będzie miał układ zbliżony do oryginalnego arkusza *Moja\_OPR\_list.xlsm*, czyli np. arkusz z listą jednostek (kolumny: nazwa, liczba modeli, jakość, obrona, wytrzymałość, zdolności, broń, koszt) oraz arkusz „Zbrojownia” z użytymi typami broni wraz z ich parametrami. Nie jest to priorytet, więc można to zaimplementować na końcu lub nawet odłożyć, jeśli PDF spełni wszystkie potrzeby. Ważne, by eksportowany PDF/Excel odzwierciedlał aktualne zasady punktowe – dlatego generowanie tych plików będzie korzystać z tych samych funkcji obliczeniowych co interfejs (żeby np. w PDF była ta sama suma punktów co w aplikacji). Poniżej przedstawiono przykładowy wygląd wydruku listy armii w stylu *„ramka”*, który aplikacja powinna osiągnąć:

*Przykładowy wygląd wydrukowanej listy armii (układ „ramka”) – każda jednostka jest prezentowana w osobnym polu z nazwą, statystykami (Jakość, Obrona, Wytrzymałość), przypisanymi zdolnościami oraz wyposażeniem wraz z kosztami punktowymi.*

* **Skalowalność na urządzeniach mobilnych:** Interfejs zostanie zaprojektowany w podejściu *mobile-first* lub przynajmniej *responsive*, co oznacza, że będzie automatycznie dostosowywać się do wielkości ekranu. Użycie Bootstrapa zapewni standardowe zachowanie, np. wielokolumnowe układy na szerokim ekranie będą składać się do jednej kolumny na wąskim ekranie telefonu[[10]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Skalowalno%C5%9B%C4%87%20do%20wy%C5%9Bwietlania%20na%20ekranie,panoramicznym%2C%20laptopie%2C%20tablecie%2C%20telefonie). Tabele z danymi, które mogą być szerokie (jak lista broni), otrzymają styl przewijania w poziomie na małych ekranach zamiast „wypychania” poza ramkę. Menu nawigacyjne zmieni się w rozwijane (tzw. hamburger) na smartfonach. Ponadto planujemy przetestować interfejs na urządzeniach dotykowych, by upewnić się, że przyciski są wystarczająco duże, a elementy interaktywne (np. pola wyboru liczby modeli) wygodne w użyciu. Dzięki temu aplikacja będzie użyteczna nie tylko na PC, ale także gdy gracz w trakcie gry zechce sprawdzić coś na telefonie lub tablecie.
* **Zgodność z zasadami *„Moje OPR”*:** Wszystkie mechanizmy aplikacji będą projektowane tak, by odzwierciedlać zasady gry opisane w dokumencie *Moje OPR.docx*. Poza kalkulacją kosztów (omówioną wcześniej), dotyczy to także przestrzegania limitów i reguł specjalnych. Przykładowo, jeśli zasady określają maksymalną liczebność oddziału czy ograniczenia co do powtarzalności jednostek, edytor listy będzie to respektował (np. można zablokować dodanie więcej niż X razy danej unikalnej jednostki do jednej listy). Mechanizmy stanu jednostek (wyczerpana, przyszpilona, zniszczona itp.) są bardziej związane z rozgrywką niż budową listy, więc nie dotyczą bezpośrednio aplikacji – aplikacja skupia się na tworzeniu list przed grą. Jednakże opisy zdolności i cech zawarte w aplikacji (np. po najechaniu na nazwę zdolności można wyświetlić podpowiedź z jej opisem) będą zgodne z zasadami (np. *Regeneracja: podczas obrony naturalna 6 pozwala zignorować ranę*[[22]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Regeneracja%3A%20Podczas%20obrony%2C%20za%20ka%C5%BCd%C4%85,ran%C4%99%20przydzielon%C4%85%20podczas%20tego%20ataku)). Upewnimy się, że każdy atrybut jednostki i broni znaczy to samo co w zasadach (Quality – test cechy modelu, Defense – wartość obrony, Tough(X) – wytrzymałość X oznacza, że model ma X ran itp.). W razie wątpliwości, przyjmiemy zasadę zdrowego rozsądku i zgodność z opisem – np. zdolność *Delikatny* obniżająca skuteczność naturalnych 6 w obronie będzie uwzględniona poprzez modyfikator w kosztach lub notkę ostrzegawczą[[23]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Delikatny%3A%20Podczas%20test%C3%B3w%20obrony%20naturalna,6%20nie%20oznacza%20automatycznego%20sukcesu). Aplikacja ma być narzędziem pomocniczym, więc kluczowe jest, by gracz przygotowujący listę mógł w pełni zaufać, że jeśli coś jest dozwolone w zasadach, to aplikacja mu na to pozwoli, a jeśli coś jest ograniczone lub kosztuje dodatkowe punkty – aplikacja to uwzględni. W tym celu będziemy ściśle trzymać się informacji z dokumentów od użytkownika podczas implementacji logiki.

Na podstawie powyższego planu, aplikacja będzie kompleksowym rozwiązaniem do tworzenia list armii i zarządzania danymi do bitewnej gry figurkowej, spełniającym wszystkie wyszczególnione wymagania. Plan uwzględnia podział na moduły, etapy implementacji ułatwiające kontrolę postępów, szczegóły techniczne dotyczące struktury i środowiska oraz sposób, w jaki każde z kluczowych wymagań zostanie zaadresowane. Dzięki temu użytkownik (inicjator projektu) może przystąpić do realizacji krok po kroku, mając pewność, że efekt końcowy będzie zgodny z założeniami opisanymi w dokumentach „Moje OPR.docx”, „MojaArmyForgr.docx” oraz odzwierciedli funkcjonalność arkusza „Moja\_OPR\_list.xlsm”.

[[1]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Mo%C5%BCliwo%C5%9B%C4%87%20zalogowania%20si%C4%99,link%20do%20aplikacji%20jest%20zaufany) [[2]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Armie%3A) [[3]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Armie%20mog%C4%85%20by%C4%87%20tworzone%20od,jednostki%20z%20odno%C5%9Bnikami%20do%20bazowych) [[4]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Przy%20tworzeniu%20sugerowane%20jest%20wybranie,armii) [[5]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Eksport%3A) [[8]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Lista%20zdolno%C5%9Bci%20wraz%20z%20opisami,i%20zasadami%20obliczania) [[10]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Skalowalno%C5%9B%C4%87%20do%20wy%C5%9Bwietlania%20na%20ekranie,panoramicznym%2C%20laptopie%2C%20tablecie%2C%20telefonie) [[11]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Strona%20g%C5%82%C3%B3wna%3A) [[12]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Lista%20broni%20%E2%80%93%20uk%C5%82ad%20podobny,jak%20we%20wzorcowym%20Excelu) [[13]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Zbrojownia%3A) [[14]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Lista%20modeli%20w%20armii%3A%20nazwa%2C,podstawowe%20atrybuty%2C%20zdolno%C5%9Bci%2C%20domy%C5%9Blna%20bro%C5%84) [[15]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Najpierw%20nag%C5%82%C3%B3wek%20armii%2C%20potem%20lista,jednostk%C4%99%20do%20armii%E2%80%9D%2C%20%E2%80%9Edodaj%20armi%C4%99%E2%80%9D) [[16]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Potem%20lista%20dodanych%20jednostek%20z,modelu%2C%20oddzia%C5%82u%2C%20broni%20i%20zdolno%C5%9Bci) [[17]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Baza%20danych%20pozwalaj%C4%85ca%20zachowa%C4%87%20dane,pomi%C4%99dzy%20sesjami) [[18]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=Hierarchiczno%C5%9B%C4%87%2Fdziedziczenie%3A%20bronie%2C%20jednostki%20i%20armie,je%20do%20rodzica%20usuwanego%20elementu) [[19]](file://file-7WdmqyfVrTpm4kBMtDJLTw#:~:text=znale%C5%BA%C4%87%20wszystkie%20dzieci%20i%20przekierowa%C4%87,je%20do%20rodzica%20usuwanego%20elementu) MojaArmyForgr.docx

<file://file-7WdmqyfVrTpm4kBMtDJLTw>

[[6]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Koszt%20modelu%20jest%20sum%C4%85%20kosztu,kosztu%20broni%20i%20kosztu%20zdolno%C5%9Bci) [[7]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Koszt%20zdolno%C5%9Bci%20pasywnych%20nale%C5%BCy%20zap%C5%82aci%C4%87,za%20ka%C5%BCdy%20punkt%20wytrzyma%C5%82o%C5%9Bci) [[9]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Transport,Lataj%C4%85cy%2C%201%2C25%20Szybki%2FZwinny%2C%201%20pozosta%C5%82e) [[20]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Znaczniki%20zdolno%C5%9Bci%3A%20Ka%C5%BCdy%20odzia%C5%82%20mo%C5%BCe,albo%20zdolno%C5%9B%C4%87%20aktywn%C4%85%2C%20albo%20aur%C4%99) [[21]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=modyfikatory%20zdolno%C5%9Bci) [[22]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Regeneracja%3A%20Podczas%20obrony%2C%20za%20ka%C5%BCd%C4%85,ran%C4%99%20przydzielon%C4%85%20podczas%20tego%20ataku) [[23]](file://file-Eyb1sBb9hcEFzQDnfHnnN2#:~:text=Delikatny%3A%20Podczas%20test%C3%B3w%20obrony%20naturalna,6%20nie%20oznacza%20automatycznego%20sukcesu) Moje OPR.docx

<file://file-Eyb1sBb9hcEFzQDnfHnnN2>