Заглавие на проекта: TaskMasterPro

Автори: [Група 13]
Дата: 29 октомври 2023 г.

Въведение

Предназначение

Системата е предназначена за организации, които искат да подобрят координацията, прозрачността и ефективността на своите проекти. TaskMasterPro осигурява инструменти за планиране, назначаване, проследяване и анализ на задачи, както и за комуникация между участниците в проекта.

Обхват

Този документ покрива архитектурната концепция и дизайн на системата, като излага основните компоненти, взаимодействия и функционалности. В него се разглеждат:

- Основни архитектурни компоненти и техните взаимоотношения.
- Технологични решения и интеграции.
- Безопасност и достъп до системата.
- Разширяемост и скалируемост на системата

Актьори

- Проектни Мениджъри: Отговорни за планирането и управлението на проектите. Използват системата за назначаване на задачи, определяне на приоритети и следене на прогреса.
- Разработчици/Изпълнители: Работят по задачите, отбелязват прогреса и комуникират промени или проблеми.
- Тестери: Използват системата за докладване на проблеми или грешки.
- Системни Администратори: Отговорни за поддържането на системата, обновленията и интеграциите.

- Заинтересовани страни: Включва клиенти или други външни участници, които могат да проследяват прогреса на проекта, но не са пряко включени в неговото изпълнение.
- Всеки актьор има различен набор от права и достъп до различни части на системата, които са определени в съответствие с техните роли и отговорности.

Използвани термини и символи

- Microservice Architecture (Микросервизна архитектура): Архитектурен стил, който структурира приложение като колекция от услуги, които са лесни за поддържане и тестване, слабо свързани, независимо разгръщаеми и организирани около бизнес възможности.
- Docker (Докер): Платформа, използвана за разработка, доставка и стартиране на приложения в контейнери. Контейнерите позволяват на програмиста да опакова приложението с всички необходими части, като библиотеки и други зависимости, и да ги изпрати като един пакет.
- Kubernetes (Кубернетис): Отворена система за автоматизация на развертане, мащабиране и управление на контейнеризирани приложения. Тя групира контейнери, които съставляват приложение, в логически единици за лесно управление и откриване.
- API (Application Programming Interface - Приложен програмен интерфейс): Комплект от правила и протоколи за създаване и взаимодействие със софтуерни приложения. API определя как трябва да взаимодействат софтуерните компоненти.
- Caching (Кеширане): Техника, използвана в компютърната наука за съхранение и повторно използване на предварително извлечени или изчислени данни, така че бъдещи заявки за същите данни да могат да бъдат обслужвани по-бързо.
- Lazy Loading (Мързеливо зареждане): Дизайнерски модел, често използван в компютърното програмиране, който отлага инициализацията на обект до момента, в който е необходим. В уеб дизайна се отнася до зареждане на определени елементи на уеб страница само когато те са необходими.
- Profiling (Профилиране): В софтуерното инженерство, профилирането е форма на динамичен анализ на програми, който измерва, например, пространствената (памет) или времевата сложност на програма.
- Redis (Редис): Проект за данни в паметта, реализиращ разпределена ключ-стойност база данни с опционална устойчивост.
- Memcached (Мемкашд): Обща система за разпределено кеширане на паметта. Често се използва за ускоряване на динамични уебсайтове, кеширащи данни и обекти в RAM.

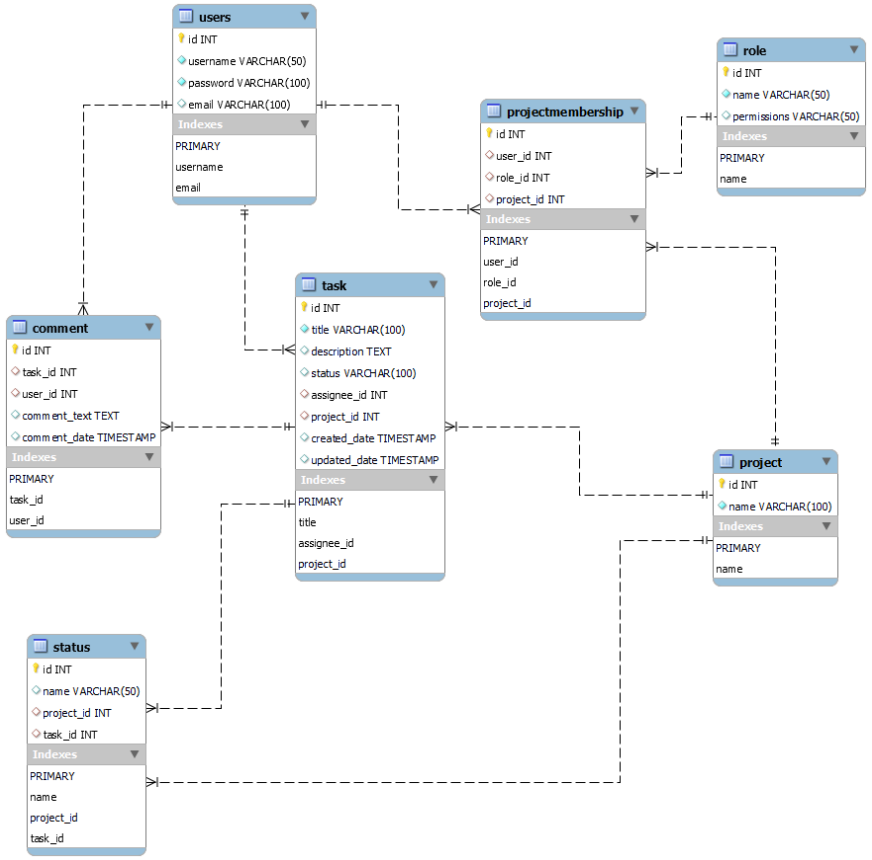
- Load Balancing (Балансиране на натоварването): Разпределение на входящия мрежов трафик между група от задни сървъри.
- Backup (Резервно копие): Процес на копиране и архивиране на компютърни данни, така че да могат да се използват за възстановяване на оригинала след загуба на данни.

Източници

ChatGpt, Medium, Stackovertflow

Архитектурен обзор

Изглед на данните



Диаграмата представя схема на база данни за система за управление на задачи. В нея са включени ключови компоненти като:

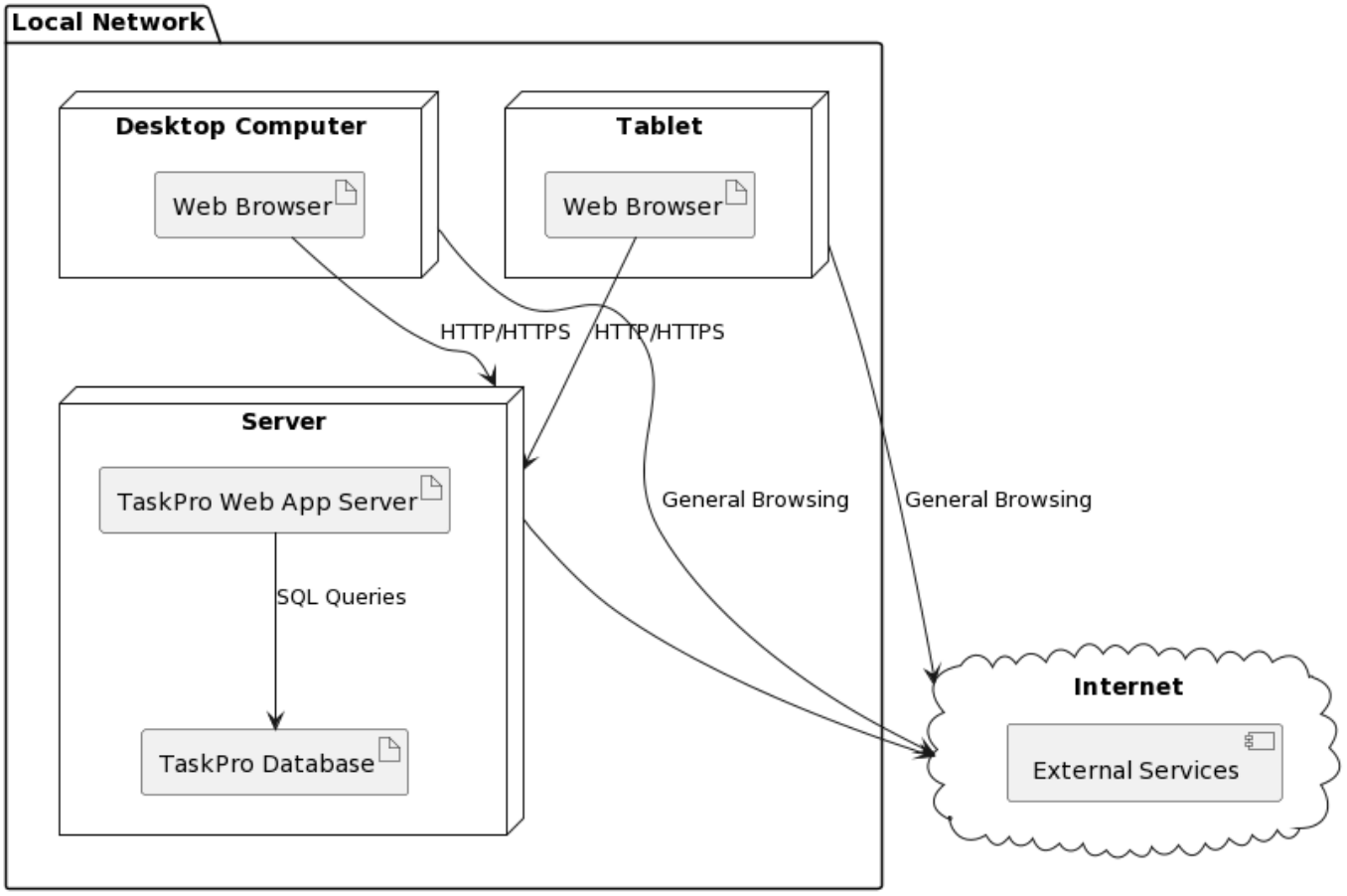
- Потребители (users): Съдържа информация за регистрираните потребители.

- Проекти (project): Дефинира различни проекти, в които могат да се създават задачи.
- Задачи (task): Представя конкретни задачи, свързани с даден проект.
- Коментари (comment): Позволява потребителите да добавят коментари към задачите.
- Роли (role): Дефинира различни роли в рамките на проекта.
- Членство в проект (projectmembership): Определя участието на потребителите в различни проекти и техните роли в тях.
- Статуси (status): Определя възможните статуси, които една задача може да има.

Тази диаграма е полезна, тъй като предоставя визуално представяне на структурата и взаимовръзките на базата данни. Тя помага на разработчиците да разберат как данните са организирани и как взаимодействат помежду си. Чрез такива диаграми можем да гарантираме правилното функциониране на системата, да идентифицираме потенциални проблеми и да оптимизираме производителността.

В съвременната софтуерна разработка, диаграмите като тази са критични, защото те осигуряват яснота и структура, които са необходими за създаване на надеждни и мащабируеми системи.

Изглед на внедряването

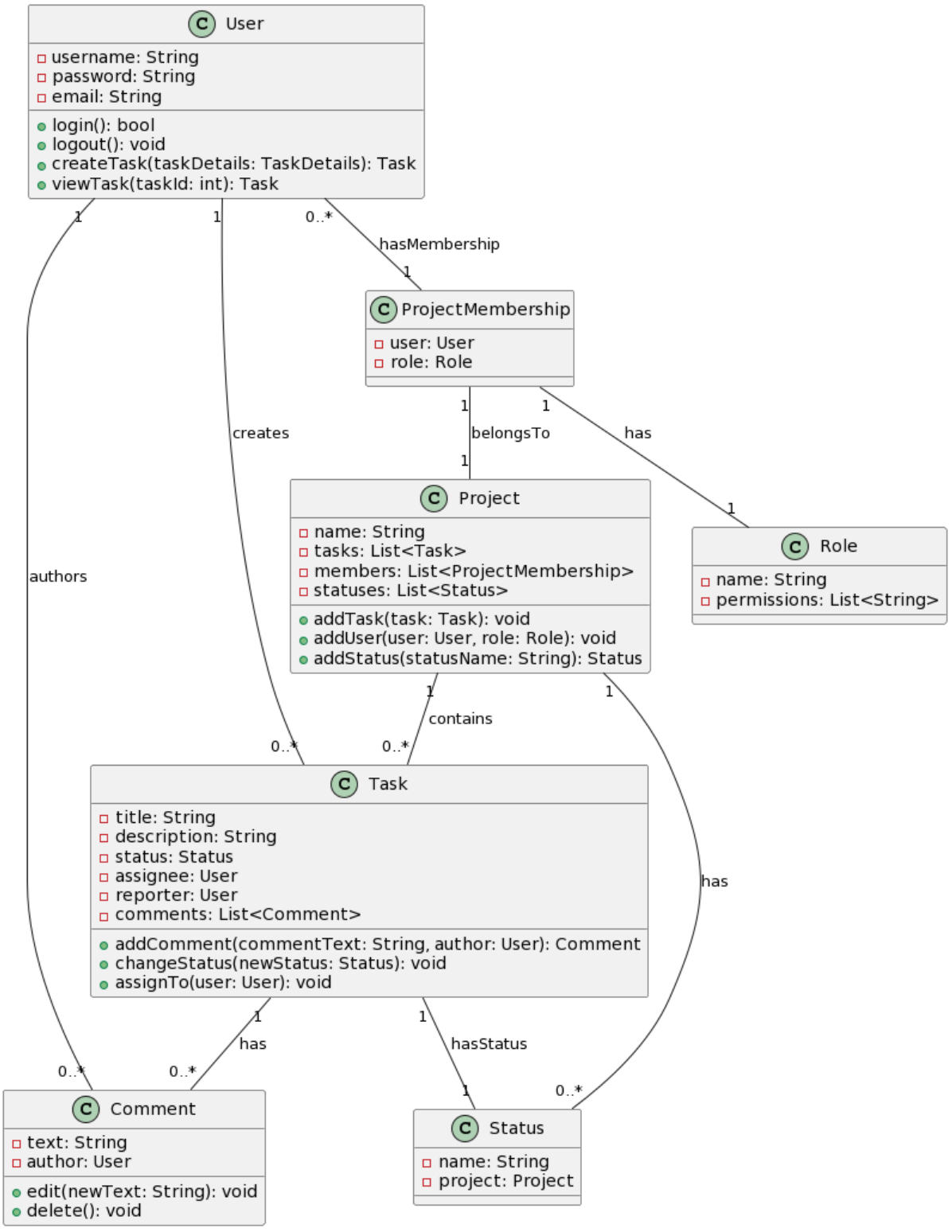


Диаграмата илюстрира мрежовата структура и комуникацията между различните компоненти на системата "TaskPro".

- Локална мрежа: Включва както десктоп компютър, така и таблет, които използват уеб браузъри за достъп до системата.
- Сървър: Съдържа приложението "TaskPro Web App Server", което обработва заявки от уеб браузъра и извършва SQL заявки към базата данни "TaskPro Database".
- Интернет: Сървърът може да комуникира с външни услуги чрез интернет, докато потребителските устройства могат да използват интернет за обща уеб навигация.

Схемата е полезна, тъй като показва как потребителите достъпват системата и как системата комуникира с външни услуги. Това осигурява яснота по отношение на трафика и комуникационните канали, което е важно за осигуряване на ефективност и сигурност на системата.

Логически изглед

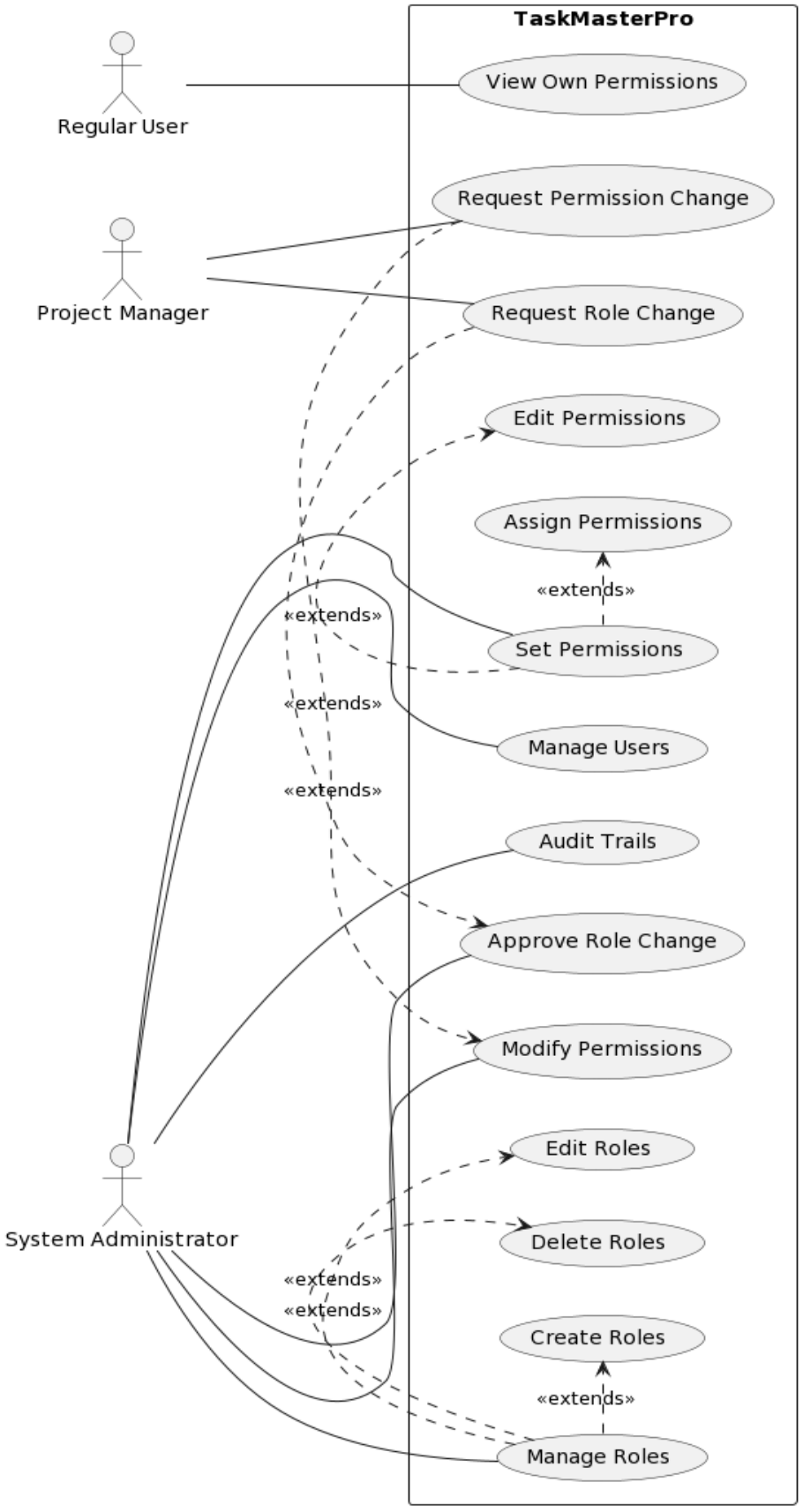


Диаграмата представя обектно-ориентиран модел на системата "TaskPro" чрез класове и техните взаимоотношения.

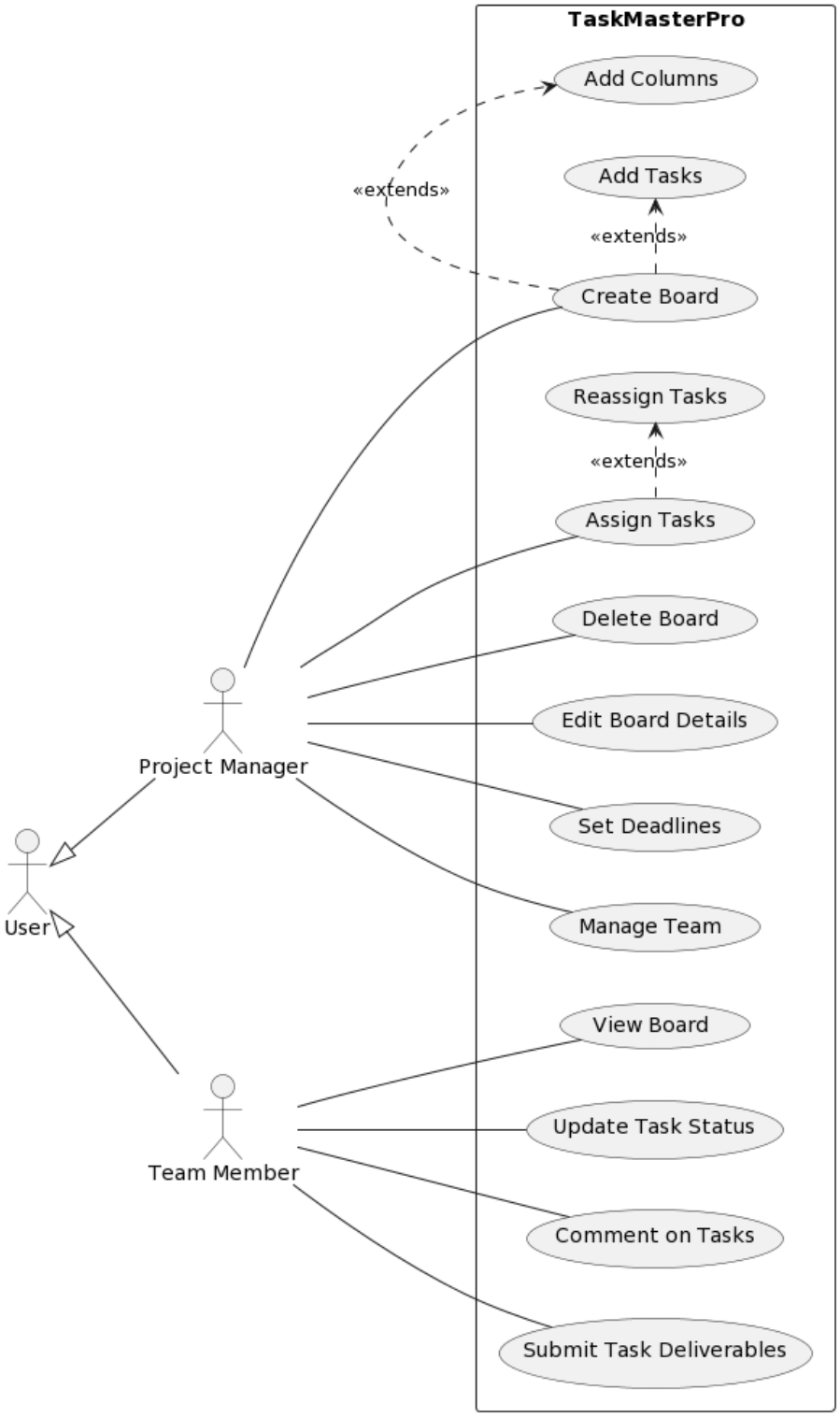
- Потребител (User): Съдържа основната информация за потребителя, както и функционалност за влизане в системата, създаване и преглед на задачи.
- Проект (Project): Описва отделен проект със свойствата име, списък с задачи, членове и статуси. Съдържа методи за добавяне на задачи, членове и статуси.
- Задача (Task): Има свойства като заглавие, описание, статус, отговорник и коментари. Съдържа функционалност за добавяне на коментари, промяна на статуса и назначаване на отговорник.
- Коментар (Comment): Включва текст на коментара и автора му, с функционалност за редактиране и изтриване.
- Статус (Status): Описва различните статуси, които може да има една задача в рамките на проекта.
- Роля (Role): Представлява различните роли, които могат да бъдат назначавани на потребителите в рамките на даден проект, със свойства като име и разрешения.
- Членство в проект (ProjectMembership): Илюстрира отношенията между потребителите и проектите, като свързва потребителя със съответната му роля в проекта.

Моделът е полезен, тъй като предоставя структуриран поглед върху основните обекти в системата и взаимодействията между тях. Това помага при проектирането и разработката на системата, осигурявайки яснота и последователност на функционалностите и действията.

Use-case изглед

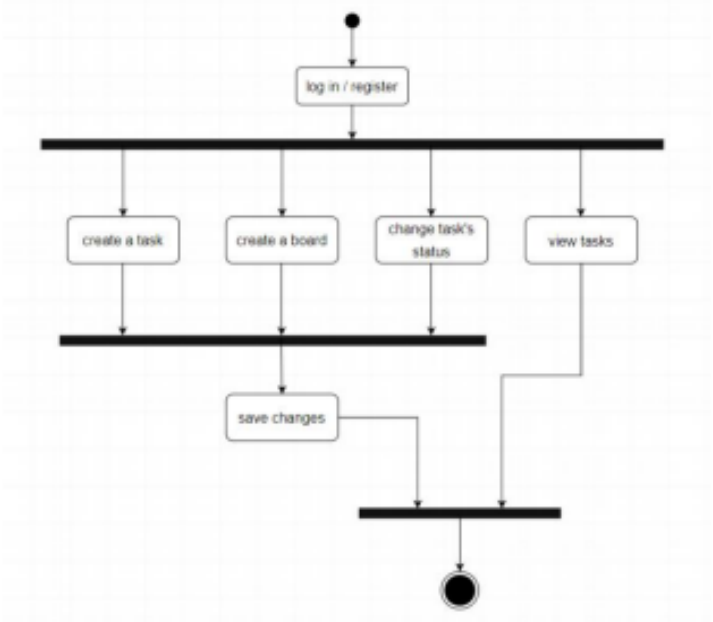


Управление на задачите



В процеса на създаване и менажиране на задачи в "TaskMasterPro" има три ключови роли: Project Manager, User и Team Member. Процесът за създаване и управление на задачи е както следва: Мениджърът на проекта има правомощията да създава табла. След като дъската е създадена, те могат да добавят колони, задачи и да управляват различни подробности за дъската.

Задачите могат да се добавят от ръководителя на проекта и след като са на борда, те могат да възлагат или превъзлагат тези задачи на конкретни членове на екипа.
Ръководителят на проекта може също така да задава срокове за изпълнение на задачите и да управлява целия екип, работещ в борда.
Членовете на екипа основно взаимодействат със задачите. Те могат да преглеждат таблото, да актуализират състояния на задачи, да коментират задачи и да изпращат резултати, свързани със задачи.
Потребител, който може да се счита за обща роля, може да преглежда дъските, като им дава представа за текущи проекти и задачи.
Тази структура гарантира ясно разграничение между управленските отговорности и ориентираните към задачи роли, улеснявайки ефективното управление на задачите и сътрудничеството.



Предоставените диаграми представят функционалностите и взаимодействията на потребителите със системата TaskMasterPro. Диаграма на Потребителските Истории (Use Case Diagram): Тя илюстрира взаимодействията между потребителите (Обикновен потребител и Тим лидер) и функционалностите на системата. Основни моменти включват:

И двамата потребители могат да приоритизират задачи, да задават крайни срокове, да променят статуса на задачите и да преглеждат задачи.
Само "Тим лидерът" може да създава дъска (борд) и да управлява достъпа на членовете на екипа.

"Обикновеният потребител" има възможности като създаване на задачи и задаване на статус на задачите.
Диаграма на Дейностите (Activity Diagram): Тя представлява последователността от действия, които потребителят изпълнява в системата:
Началото е с вход или регистрация на потребителя.
След това потребителите могат да създават задача, да създават дъска (борд), да променят статуса на задача или да преглеждат задачи.
Промените, направени от потребителя, се запазват преди завършване на сесията.
Тези диаграми са съществени, тъй като предоставят визуално представяне на взаимодействията на потребителите и функционалностите на системата, което помага за разбирането на поведението на системата и потенциалните процеси по разработка на софтуера.

Нефункционални изисквания

Достъпност

1. Откриване на проблеми:
    - Интеграция с инструменти: Системата “taskpro” се интегрира с различни инструменти за мониторинг като Grafana, Prometheus и New Relic, за да предоставя по-голям контрол на работоспособността.
    - Алармиране: При откриване на нестандартно поведение или изпадане извън установените параметри, системата автоматично изпраща сигнали до администраторите.
    - Анализ на логове: Логовете се анализират регулярно с помощта на инструменти като ELK stack (Elasticsearch, Logstash, Kibana), за да се открият и диагностицират потенциални проблеми.
2. Възстановяване:
    - Автоматизирани бекъпи: Освен репликациите, системата автоматично прави резервни копия на данните на различни интервали, което улеснява възстановяването.
    - Точки на възстановяване: За осигуряване на максимална защита, създават се точки на възстановяване, които позволяват на системата да се върне към предишно състояние.
    - • Хоризонтално мащабиране: В случай на нужда от повече ресурси или при отказ, нови инстанции на приложението може да бъдат автоматично стартирани в облачната среда.
3. Превенция:

- Сканиране за уязвимости: Инструменти като OWASP ZAP и SonarQube се използват регулярно за проверка на кода и инфраструктурата за потенциални уязвимости.

Разширяемост

1. Архитектурни шаблони:
    - Декомпозиция на услугите: Чрез използване на микросервизи, системата може да разделя функционалностите си на отделни, леки компоненти, което улеснява разширяването.
    - Оркестрация на услуги: Микросервизите могат да комуникират помежду си, използвайки съобщения или API заявки, което осигурява гъвкавост в разработката и разширяването.
    - Независимост от платформата: Микросервизите могат да бъдат разработени и развертани независимо от езика или платформата, която използват другите части на системата.
2. Технологии:
    - Скриптове за развертане: С Docker и Kubernetes може да се автоматизира развертането на нови версии или услуги, без да е необходимо ръчно намешателство.
    - Мащабиране на ресурсите: Kubernetes осигурява автоматично хоризонтално и вертикално мащабиране, базирано на натоварването на системата.
    - Балансиране на натоварването: Използвайки интегрирани инструменти за балансиране на натоварването, системата може да управлява голямо количество заявки без проблеми.
3. Похвати на проектиране:
    - Стандартизирани API протоколи: Използвайки стандартни протоколи като REST или GraphQL, системата може лесно да комуникира с външни приложения или платформи.
    - API Gateway: Централизираната точка за управление на всички API заявки улеснява мониторинга, сигурността и кеширането.
    - Плъгин-базирана архитектура: Позволява добавянето на нови функционалности чрез плъгини или модули, без да се налага промяна в основния код на системата.

Производителност

1. Ограничаване на натоварването:
    - Компресия на данни: Използването на методи за компресия може да намали обема на пренасяните данни и да ускори времето за реакция.

- Оптимизация на изображения и медийни файлове: Уменьшаването на размера на файлове може значително да ускори времето за зареждане на страницата.
- Lazy Loading: Тази техника позволява на страницата да зареди само необходимите ресурси, когато са необходими, вместо да зарежда всичко едновременно.

2. Планиране:
    - Профилиране: Регулярно профилиране на кода може да идентифицира участъци, които са интензивни по отношение на ресурсите и изискват оптимизация.
    - Автоматично мащабиране: Системата може да бъде настроена да разпределя ресурсите, когато те станат недостатъчни и да ги освобождава, когато натоварването намалее.
    - Тестове за производителност: Редовно изпълнение на тестове може да помогне да се гарантира, че системата постоянно отговаря на изискванията за производителност.

3. Репликация:
    - Разпределено кеширане: Използването на разпределени кеш системи, като Redis или Memcached, може да осигури бърз достъп до често използваните данни.
    - Балансиране на натоварването: Пренасочване на потребителските заявки към наличните ресурси, за да се гарантира равномерно разпределение на натоварването.
    - Резервни копия на данни: Редовното правене на резервни копия и географско разпределение на тях може да предостави допълнителна защита и бърз достъп в случай на проблеми.

Сигурност

- Ролева система: Подробна система за управление на ролите позволява на всеки потребител да има индивидуален набор от права, базирани на неговите отговорности и нужди.
- Ограничаване на достъпа: Всеки опит за достъп извън дефинираните правила се регистрира и анализира. Неправомерният достъп се блокира автоматично.
- Резервни копия: Автоматизирани резервни копия се съхраняват в различни географски региони, за да се гарантира целостта на данните дори и при катастрофални събития.

Използваемост

Определянето на използваемостта се състои именно в това, от какво ниво на обучение има нужда потребителят, за да изпълни плановете и целта си в приложението. Въпреки че е свързано повече с дизайна отколкото с архитектурните характеристики, използваемостта не е по-малко важна. Едни от основните фактори за тази характеристика, които ще покрием, са:

- изработване на интуитивен дизайн, който да съкрати нужното на потребителя време за разбиране на архитектурата и навигацията на системата
- улесняване на използването на приложението, за да може колкото се може по-бързо разработчик, без предишен досег с него, да се научи да го използва
- ефективност на използването, което се определя от това колко време би отнело на вече опитен разработчик да изпълни целите си в приложението
- запомняемост, което означава, че и при следващото си посещение потребителят ще е запомнил достатъчно, за да го използва ефективно пак
- намаляване честота и сериозността на грешките, които потребителят може да допусне в използването на приложението

Един от начините, с които ще си послужим, за да са налице по-горните фактори, се състои в използването и “имитирането” на заобикалящия ни свят в концепциите, иконите и т.н. на приложението. Също така ще са на лице съобщения и нотификации, които съдържат описание на едно единствено действие (а не няколко действия описани в едно съобщение) и в това описание ще липсват жаргонни думи. Друг такъв начин е задаване на лимит на възможните опции, за да се избегне затрупване на дисплея и потребителя с информация наведнъж (т.е. показва се само информацията от първа необходимост, нужна за изпълнение на задачата).
Използването на правилен размер, цвят и шрифт на текста е от изключителна важност, понеже така ще бъде изградена ясна йерархия на достигащата до потребителя логическа информация и ще се осигури по-добра четимост и организираност. Включените предупреждения и автокоректор ще намалят честотата на допускани от потребителя грешки, а в случай на все пак допуснати такива ще бъде осигурен и бутон за връщане една стъпка назад. В основата на системата ни седи именно това, че потребителя може да създава собствените си задачи, бордове, да променя статусите и т.н., което с други думи означава, че приложението ни предлага персонализиране, правейки го още по-лесно за работа за разработчика, който собственоръчно създава различни компоненти.

Възможност за тестване

Възможност за тестване: Гарантирането на здравината и надеждността на TaskMasterPro е от огромно значение за нашия екип. Като система за управление на задачите, TaskmasterPro се задължава да работи безупречно за да подсигури на екипите, които го ползват лесен и безгрешен процес на работа. Тестването в различните му форми ни позволява да оценим, усъвършенстваме и валидираме функционалността на софтуера на различни нива. От детайлните компоненти до по-широките взаимодействия и цялостната производителност на системата, ние сме ангажирани със строг режим на тестване. По-долу е описан нашият структуриран подход към трите основни нива на тестване: ”unit”, интеграция и система.

Unit тестване - Свързано с гарантиране, че отделните компоненти или единици на TaskMasterPro ще функционират правилно, когато биват изолирани.

- Компоненти за тестване: В софтуера за управление на задачи единиците обикновено включват:
    - Механизми за създаване, редактиране и изтриване на задачи.
    - Услуги за уведомяване (известия).
    - Функции за управление на потребителите (регистрация, влизане, излизане).
    - Потребителски настройки и предпочитания.
- Имплементация:
    - Ще реализираме нашите тестове с помощта на JUnit, като се има предвид, че нашата база данни е базирана на Java.
    - Ще напишем тестове, които покриват както „щастливия път“ (очаквано поведение), така и „крайни случаи“ (потенциални области, където нещата могат да се счупят или където можем да получим неочаквано въвеждане от страна на потребителя).
    - За да гарантираме, че нашите тестове са точни, ние използваме “mock up” на външни услуги или зависимости, което ни позволява да се съсредоточим единствено върху unit-ите, които изследваме.

Тестване на интеграцията - За интеграционното тестване ще възприемем систематичен подход. Като се има предвид, че различните компоненти на TaskMasterPro трябва да работят в тандем, ще се съсредоточим върху това как те взаимодействат.

1. Интеграционни точки: Ще разгледаме критичните точки, в които комуникират различни модули на TaskMasterPro. Това може да е мястото, където модулът за управление на потребителите взаимодейства с управлението на задачи или как нашият бекенд взаимодейства с нашата база данни с помощта на JUnit за Java.

2. Методология на тестването: Ще дадем приоритет на реалните взаимодействия пред симулираните. Въпреки това, когато взаимодействията в реално време с външни услуги са непрактични или изискват много ресурси, можем да ги симулираме или да използваме “mock up”, за да възпроизведем потенциални сценарии и отговори от реалния свят. Това ще гарантира, че не сме зависим единствено от наличността или надеждността на външни услуги по време на нашите тестове.

3. Сценарии за покриване: Ние ще се стремим да валидираме целия работен процес. От регистриране на потребител, управление на задачи, получаване на известия до настройване на настройките му, ние ще гарантираме гладко взаимодействие на всяка стъпка.

4. Инструменти и симулации: За тестване на API взаимодействия обмисляме инструменти като Postman.

5. Очакван резултат: в края на фазата за тестване на интеграцията нашата цел е компонентите на TaskMasterPro да си взаимодействат безпроблемно, без спънки или несъответствия в данните.

Тестване на системата - Главната цел на екипа на TaskMasterPro е да потвърди, че нашият софтуер, разглеждан като цялостна система, работи безупречно и отговаря на нашите предварително определени изисквания.

1. Обхват на системното тестване: Системното тестване за нас е валидиране на цялото приложение в среда, която отразява условията в реалния свят.

2. Тестови сценарии и use cases: Ще преминем през целия път на потребителя. От момента, в който потребителят се регистрира, създава, управлява или изтрива задачи, получава известия, до момента, когато коригира настройките на акаунта си или дори излезе, ние ще проверяваме внимателно всяко взаимодействие.

3. Среда: Решаващ аспект от нашето системно тестване е средата. Ние ще гарантираме, че нашата производствена среда бива възможно най - близко репликирана, възпроизвеждайки потенциални предизвикателства в реалния свят. Независимо дали става въпрос за различни устройства, браузъри или операционни системи, ние искаме TaskMasterPro да бъде стабилен навсякъде.

4. Производителност и натоварване: Длъжни сме да симулираме как TaskmasterPro би се държал под високо натоварване. Планираме да симулираме големи потребителски натоварвания, пикови времена и други сценарии на стрес, за да преценим времето за реакция и стабилността на системата.

5. Обратна връзка: По време на фазата за тестване целия екип ще си комуникира постоянно и ще подсигури оправянето на бъгове, независимо колко незначителни са те.

6. Очакван резултат: В заключение на нашето системно тестване нашият стремеж е ясен: TaskMasterPro трябва не само да изпълни, но и да надхвърли първоначалните ни изисквания. Потребителите трябва да могат да навигират и да използват всяка функция без проблеми, независимо от тяхното устройство или платформа.

Интероперабилност

- Стандартизация: За да се осигури лесна интеграция и съвместимост, системата поддържа универсални стандарти и протоколи.
- API Интеграции: Отвореният API предоставя възможност за взаимодействие с различни външни системи, приложения и платформи, което улеснява обмена на информация.
- Документация: Ясна и подробна документация на интерфейсите и API-тата улеснява разработчиците при интеграционни проекти.
- Адаптивност: Системата е дизайнирана да приема и обработва данни от различни формати и източници, като автоматично преобразува информацията, за да бъде съвместима с вътрешните структури.
- Конвертиране и мапинг: Вградените инструменти позволяват автоматично конвертиране на данни и мапинг между различни формати и структури.
- Съвместимост назад: При актуализации или промени, системата запазва съвместимостта с предишни версии, за да се избегнат

потенциални проблеми при интеграция със стари системи или платформи.