Софийски университет „Св. Кл. Охридски”
Факултет по математика и информатика
Катедра „Софтуерни технологии”

ДИПЛОМНА РАБОТА

на тема

„Система за синтез и визуализация на текст с разнообразен произход (Ultimate Speaker) „

Дипломант: **Асен Георгиев Асенов**
Специалност: **Разпределени системи и мобилни технологии**
Факултетен номер: **M24072**

Научен ръководител:
доц. д-р Милен Петров

София, 2015г.

Съдържание

Глава 1. Увод

Дипломната работа има за цел разработването на уеб базирана система, предоставяща неограничени възможности на потребителите, желаещи да трансформират разнообразен набор от текстови във аудио файлове и презентации във виртуални такива, с вграден звук. Основните функции на системата са:

- Четене на *HTML*, *TXT*, *PDF*, *DOC* файлове, генериране и възпроизвеждане на аудио записи (*WAV*) от прочетеното.
- Разчитане на *PPT*, *PPTX* презентации и визуализирането им в Уеб вариант, с вграден прочит на страниците.
- Добавяне на рейтинг към конкретен прочит.
- Предоставяне на възможност за търсене във файловете, намиращи се в системата.
- Поддръжка на потребителски акаунти – регистрация, вход, изход, споделяне на файлове.
- Всички операции се изпълняват посредством *Restfull* заявки към сървъра.

Продукта ще бъде разширим, позволявайки обогатяване, чрез добавянето на допълнителни файлови формати, гласове за четене, варианти за визуализиране на презентационните документи, възможност за командване на системата чрез глас (*SpeechToText*), което на практика прави възможностите му неограничени. За реализиране на системата трябва да се използват технологии и езици за програмиране, които са платформено независими, като така ще се премахне ограничението да се използва само конкретна операционна система, което дава допълнителна свобода на потребителите. За съхранение на системни настройки, файловете за прочит, генерираните аудио файлове и променените презентации ще се ползва база от данни. За визуализация на презентациите ще се използва готово решение, което да позволява лесна интеграция и посредством което ще се улесни разработването на системата. За четене на различни видове текст ще се използват готови синтезатори на глас, които трябва да са безплатни и да работят на всички операционни системи.

1.1. Актуалност на проблема и мотивация

За момента съществуват няколко подобни продукта, предоставящи четене на *pdf*, *doc*, *ppt* файлове. Основните недостатъци на тези системи са:

1. Голяма част от тях са платени и използването им е ограничено, което представлява трудност за крайния потребител
2. Всеки продукт поддържа определени файлови формати, което означава че ако искаме да използваме широк набор от файлове, ще се наложи инсталирането на повече от един продукт (евентуално заплащането на допълнителни лицензи). Също така всяка система разполага със собствен глас за четене и е възможно да чуваме разлики в прочита на отделните

файлове. Всичко това представлява затруднение за крайния потребител.

3. Голяма част от тези системи не поддържат български език.

Съществуването на много продукти, предоставящи възможност за трансформиране на текст във звук, ни навежда на факта, че има потребителски интерес. Това е добър начален стимул, като идеята е да се избегнат всички недостатъци на подобни продукти, за да се създаде конкурентна система, която да превъзхожда останалите. Създаването на безплатна система, ще разшири кръга на евентуалните потребители, тъй като няма да има финансови ограничения. Така има възможност продукта да се популяризира сред учебните институции, с цел по-бързо и лесно научаване на конкретен въпрос. Друга алтернатива е популяризирането му сред хората с увреждания, за които четенето е физически невъзможно. Друго предимство е възможността за разширяване на продукта, с което на потребителя се дава свобода да използва и променя програмата, като така я адаптира за личните си нужди. Множеството приложни области, са добра предпоставка за бързо популяризиране и разрастване на продукта.

Възможността за създаване на система, представляваща цялостно безплатно решение, което да превъзхожда останалите и да предоставя на потребителя неограничени възможности, мотивира започването на разработката.

1.2. Цел и задачи на дипломната работа

Ultimate Speaker има за цел да предостави на потребителите лесен и удобен начин за генериране на прочит и визуализация на разнородни файлове. Системата може да се използва от потребители с увреждания или просто от хора, които не могат да отделят много време за стоене пред компютъра. Запис от прочита може да бъде прослушван докато пътувате, работите, разхождате се в парка и всичко което ще ви е необходимо е обикновен *MP3* плеър. Така ще се улесни възприемането, а ученето на лекции и материали ще се превърне в приятно занимание.

За да се създаде описаната система е нужно да се изпълнят следните задачи:

1. Избор на инструмент за синтез на глас. Той трябва да е с безплатен лиценз и да е платформено независим. Поддръжката на голям брой езици е предимство. Трябва да се разгледат характеристиките на различните гласове – като интонация, произношение и т.н.

2. Подбор на подходящи инструменти за разработване на проекта. Езика за програмиране на сървърната част ще е *Java* (*JDK* 7), среда за разработка *Eclipse* и система за контрол на файловете *GIT*. Подбор на система за компилиране и пакетиране на готовия продукт, операционна система за разработване.

3. Избор на допълнителни продукти, нужни за съхранение на потребителските данни и сървърните настройки, за разработване на клиентската част, за визуализация на качените и обработени презентации и др.

4. Изграждане на бизнес логиката на продукта, използвайки софтуерни

шаблони за програмиране, с цел улесняване поддръжката и разширяването на системата.

5. Избор и интегриране на инструменти за извличане на текст от разнородни видове файлове *HTML*, *PDF*, *DOC* и *PPT*. Възможността за разширяване на поддържаните файлове е предимство.
6. Разработване на *Rest API*, през което да се осъществяват всички действия с програмата. Избор на инструмент, който да поддържа *JAX-RS* стандарта за разработване на *Restfull* уеб услуги. Използването на тази технология ще премахне ограничението за използване на конкретен клиентски софтуер, което прави системата по-гъвкава.
7. Тестване на системата по време на разработка (*Unit Testing*). Създаване на автоматични тестове, които да се изпълняват при всяко компилиране и пакетиране на готовия продукт, с цел своевременно откриване на тривиални проблеми.
8. Автоматизиране инсталирането и конфигурирането на системата и използваните допълнителни продукти. Създаване на платформено зависими инсталатор, ако е необходимо, които да изпълняват специфичните за всяка операционна система операции.
9. Изпълнение на различни видове тестове на готовото приложение, под различни операционни системи, с различни по обем и вид файлове, за продължителен период от време и т.н.
10. Документиране на системата – създаване на различни видове документации, предназначени както за обикновения потребител, така и за хората, които ще искат да разширят и обогатят системата. Документиране на поддържаните операции чрез *Restfull* заявки.

1.3. Очаквани ползи от реализацията

Изграждането на безплатна система, за прочит и визуализиране на разнородни файлове, има перспектива за бързо популяризиране. Евентуалният интерес може да бъде разгледан в три основни насоки.

Първата от тях е използването на продукта в образователните институции, за подпомагане разбирането и запомнянето на определено количество информация. Тъй като голяма част от преподаватели се ориентират към електронното преподаване и информацията се предоставя както вербално така и на електронен носител, това предразполага последващото и използване в *Ultimate Speaker*. Прослушвайки генерирания прочит, на електронния урок, представлява нов начин за учене – наподобяващ разказването на даден урок от учител/професор по време на учебно занятие и предполага естественото възприемане на информацията от обучаващия се. Така на потребителя се дава възможност още веднъж да пресъздаде учебния час и така по-лесно да запомни нужната информация. Същото важи и ако сте пропуснали дадено занятие, и искате да го възпроизведете на ново. Друг полезен пример за използване на продукта е, когато сте заети с операция, ангажира зрението ви и не можете да използвате времето за учене. Прослушване прочита на желаната информация, може да

помогне в такива ситуации, като така ще отделяте по-голяма част от времето си за учене, което ще благоприятства по-бързото научаване и развитие. По-качественото обучение е една от основните ползи, очаквана след реализирането на продукта.

Втората насока, в която може да се използва системата е при хора с увреждания. Тъй като съществуват много безплатни приложения, подпомагащи незрящите при работата им с компютри, не е много сигурно колко популярен може да стане продукта в тези среди. Като основно предимство може да се разгледа възможността за прочит на огромен брой разнородни файлове, които други подобни системи може да не поддържат. Разбира се тук не можем да използваме като предимство еднаквия глас за всички прочити, тъй като системата за прочит на работния плот, ще използва друг глас и това може да създаде неудобство на потребителя. Използването на системата в тази сфера е възможно, като правилното популяризиране ще е ключово за успеха в тази насока.

Последното направление, в което се очаква програмата да придобие популярност, е използването и от обикновени потребители. Тъй като те са най-многобройна част в Интернет пространството, се очаква сравнително по-голям интерес. Всеки потребител би искал да спести време от безкрайното седене пред компютъра или електронния четец, в четене на различни видове електронна информация или книги. Това отнема както част от времето, в което може да се вършат други неща, но също така уморява очите, което води до последваща физическа умора и главоболие. Използването на системата позволява прослушване на желаната електронна информация, докато карате кола, разхождате се в парка и какво ли още не, като така се спестяват часове, които могат да бъдат отделени за любимо занимание, а в същото време не се лишавате от новата информация, която сте искали да прочетете.

Трите възможности, за реализиране на ползи от проекта, са предпоставка за успех на продукта, тъй като дори една или две от тях да се провалят, все пак ще има сфера, в която системата ще придобие популярност, а от там може да се направи опит за развитие в останалите насоки. Като цяло системата има за цел да даде възможност на потребителите да отделят допълнително време за възприемане на нова информация, като същевременно извършват неангажиращи съзнанието задачи. Така потребителя може да оптимизира свободното си време - задача, която всеки от нас се опитва да изпълни.

1.4. Структура на дипломната работа

Дипломната работа е разделена на няколко основни части, накратко обяснени по-долу:

- **Увод** – кратко описание на разработваната система и проблема, който тя решава. Дефиниране на стъпките, които трябва да бъдат изпълнени, за реализирането на програмата. Обобщаване на очакваните резултати след завършване и предоставяне системата на крайните потребители.
- **Преглед на системи и инструменти за синтез и анализ на учебни материали –** разглеждане на основните дефиниции и термини при

изграждането на дистрибутирани системи, от тип клиент-сървър, използващи „*TextToSpeech*“ технологията. Разглеждане и сравнение на съществуващи решения. Изваждане на изводи от събраната информация.

- **Използвани технологии –** описание на изискванията към използваните средства за практическото решаване на проблема. Преглед и сравнителен анализ на технологиите и платформите, които ще се използват от системата. Резултат и извод от направения избор.
- **Анализ –** описание и структуриране на функционални и нефункционални изисквания към продукта. Разглеждане на концептуалния модел на системата и бизнес процесите в нея.
- **Проектиране –** представяне архитектурата на системата и модела на съхраняваната информация. Оформяне на диаграми и поведенчески модели. Представяне на потребителския интерфейс и спомагателните модули.
- **Реализация, тестване и внедряване –** реализиране на отделните модули, интегриране с различни операционни системи, планиране и създаване на тестови сценарии, анализ на резултатите от тестовете и внедряване на готовата система.
- **Заключение –** обобщение на изпълнените задачи и цели. Насоки за бъдещо развитие, разширяване и усъвършенстване на продукта.
- **Използвана литература –** списък с използваната литература и ресурси, през целия процес по създаване на дипломната работа.

Глава 2. Преглед на системи и инструменти за синтез и анализ на учебни материали

Ultimate Speaker е система от типа клиент сървър, която може да се причисли към класа „*TextToSpeech* (*TTS*)“ програми. Най-просто те получават текст в електронен вариант и на база полученото, генерират прочит. Сървърната част позволява разширяване до дистрибутиран вид, което предполага възможност за по-голяма натовареност, с добавянето на нови компютри в групата на сървърите.

2.1. Основни дефиниции

***TextToSpeech* (*TTS*) –** това са клас от приложения за синтезиране на глас, използвани за създаване на прочит на текста от електронен документ. Създадения прочит предоставя възможност за четене на хората със зрителни увреждания или просто може да бъде използван като алтернатива на обикновеното четене. Програмите от този клас обикновено се използват заедно с приложенията за разпознаване на глас – *Voice recognition* (*SpeechToText*).

***SpeechToText* (*voice recognition*) –** програмите от този тип обработват звуков запис и го трансформират в текст. Тези програми са особено полезни при генериране на огромно количество текст в електронен формат, без много време прекарано в писане. Използват се още и от хората, които имат физически затруднения да използват клавиатура.

Дистрибутирана система – това е система която позволява увеличаване на работния капацитет, чрез добавяне на нови производствени мощности (допълнително компютри). Предимството на този вид програми е практически неограниченото увеличаване на евентуалната натовареност, което прави приложението изключително гъвкаво и адаптивно към клиентските изисквания.

Извличане на текст (*Content extraction*) – процеса на извличане на информацията, съдържаща се в определен тип файл и представянето и като обикновен текст.

2.2. Подходи и методи за решаване на проблемите

Ще разгледаме някои от основните проблеми, които възникват при разработването на системи за трансформиране на текст към звук. Всеки от тях има няколко възможни решения, които ще анализираме и ще подберем най-подходящите, за разработване на възможно най-иновативна и конкурентна програма.

2.2.1 Синтез на глас

Основен проблем при генерирането на прочит на конкретен файл е избора на синтезатора на глас. Единият вариант е да напишем такъв сами, но това само по себе си е продължителен процес, включващ както компютърни така и физични умения, и може да представлява отделна дипломна работа. Нашата задача тук не е да разработваме подобен синтезатор и за това минаваме на второто възможно решение – използване на готов инструмент, предоставящ желаните функционалности. Голяма част от синтезаторите предлагани на пазара имат

безплатна версия за хора с увреждания или за лично ползване, но използването им в нашата програма е възможно единствено след закупуване на лиценз. Те имат предимство с по-високото си качество на генерирания глас (интонация, произношение, гладкост на четене и т.н.). Хубавата новина е, че съществуват и безплатни инструменти, които стават все по-конкурентно способни и използването им набира популярност. Един от най-известните такива е синтезатора на операционната система *Windows*, който предоставя на потребителя възможност за избор на глас и конфигурирането му. Проблема при този синтезатор е че е платформено зависим и използването му през друга програма включва извикване на системни команди. Но съществуват и други безплатни инструменти, от които може да се подбере такъв, който да отговаря на изискванията за платформена независимост, възможност за конфигуриране, качество на синтезирания глас и д.р.

След решаването на този проблем, на дневен ред идва следващия – а именно какво точно ще синтезираме.

2.2.2 Извличане на информация

Извличането на информация е огромен дял в компютърната наука, занимаващ се основно с подбиране на стойностната част от даден документ и индексирането и за последващо търсене и сортиране. В това се включва сваляне на документа, извличане на съдържанието му, отделяне на важната част от това съдържания и последващото и обработване. Ние се интересуваме основно от частта с извличане съдържанието на документа. Това е процес, който е специфичен за всеки отделен тип документ – *PDF*, *DOC*, *XML*, *TXT*, *PPT* и други. От колкото повече типове документи успеем да извлечем информация, толкова по-добре ще е за нашето приложение, тъй като ще задоволи нуждите на по-голям брой потребители. Тъй като това е популярна операция, съществуват множество библиотеки, позволяващи извличането на информацията от конкретен файл. Проблема е, че повечето от тях предоставят извличане на информация само от конкретен тип документи – от *PDF*. (*Apache PDFBox*, *XpdfText*), от *DOC* (*DocRipper*), *PPT*(*catppt*). Съществуват решения, поддържащи голяма част от популярните типове документи - *Apache POI*, *Apache Tika*, *textract*, *IDOL OnDemand text extraction API*. Избора на инструмент, поддържащ повече файлови формати, качество на процеса по извличане на текста и възможност за разширяване е ключово за успеха на разработваното приложение.

2.2.3 Достъп на крайния потребител

Системата е от типа клиент-сървър, от където възниква въпроса как точно клиента ще се свързва със сървъра или по какъв начин сървъра ще предоставя достъп до поддържаните функционалности.

Първият и най-лесен вариант е сървъра да се инсталира директно на клиентската машина и да работи заедно с клиентска част. Така комуникацията може да е директна, тъй като двете програми ще работят в общ процес и няма да се изисква допълнителна организация на комуникацията. Този подход внася ограничение във възможността за поддържане на повече потребители и такива с отдалечен достъп до машината.

Вторият и най-често използван подход е инсталирането на сървърната част на публичен уеб сървър и предоставяне достъп на потребителите чрез уеб браузър. Това е доста удобно и изисква възможно най-малко от крайния потребител. Именно това е причината голяма част от съществуващите системи да използват този вариант. Това, което се идентифицира като основен недостатък при този вариант е голямото ограничение, което се налага на крайния потребител. Той няма възможност да разшири системата или да напише друг клиент, по-удобен за него. Като цяло потребителя няма никакъв друг достъп, освен предоставения му от уеб браузъра.

Третия вариант е предоставяне на достъп чрез уеб услуги. Тук съществуват много варианти, тъй като това е доста популярна сфера и технологиите които могат да бъдат използвани постоянно се увеличават. Най-широко използваните уеб услуги са *SOAP (Simple Object Access Protocol) Web Services* и *Restfull Web Services*. Като *SOAP* е остаряваща технология и все по-често бива пренаписвана с цел използването на *Restfull*.

2.2.4 Натовареност и скалируемост

Всяка система е до някаква степен ограничена от ресурсите, които са и предоставени, което налага ограничение във възможната натовареност и количеството информация което може да обработва паралелно. Съществуват два основни метода за добавяне на допълнителна изчислителна мощ и увеличаване възможностите на системата.

Първият от тях е вертикално скалируема система (*scale up*). При нея се подобряват възможностите на системата чрез добавяне на повече ресурси – *RAM*, *CPU*. Проблема при този тип системи е, че този процес е краен, тъй като количеството *RAM* и производителността на процесора на системата са ограничени и не можем да ги увеличаваме постоянно.

Вторият начин за увеличаване на производствения потенциал на програмата е разработването и като хоризонтално скалируема система (*scale out*). Тук за обработване на по-голямо количество информация, добавяме нов компютър, който да комуникира със съществуващите и да вземе от тях част от работата. При този тип системи практически няма ограничение на броя компютри, които можем да добавяме. Единствения недостатък е допълнителното натоварване при комуникацията между отделните компютри и разпределянето на задачите. Но това натоварване е незначително в сравнение с ползата, от добавянето на нова изчислителна машина.

2.2.5 Търсене

С разрастването на системата, тя ще започне да съхранява хиляди файлове и намирането на конкретен такъв започва да става трудоемка операция. Тук на помощ идва възможността за търсене в имената или съдържанието на документите. Съществуват множество библиотеки, създадени специално за оптимизиране на процеса на търсене и лесното интегриране с други системи. Най-популярните от тях са - *Apache Lucene*, *Apache SOLR Search*, *Apache Eleastic Search*. Основните разлики са годината на създаване и сложността при използването им. Също така има значение дали системата е дистрибутирана или

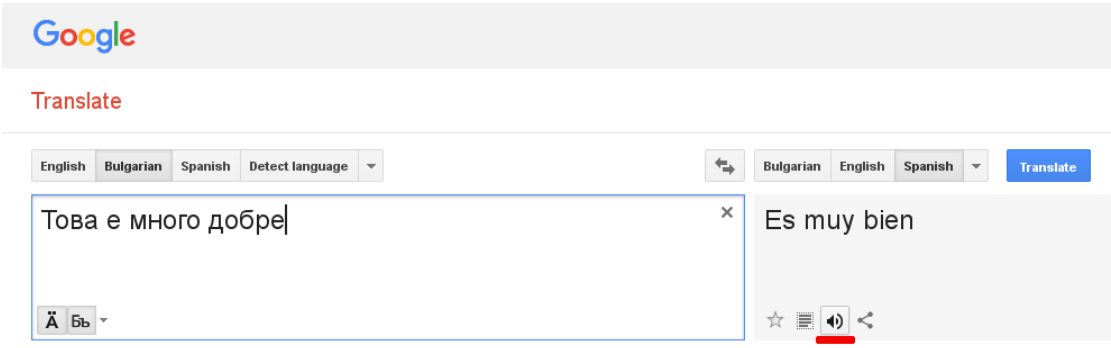
работеща само на един компютър.

2.3. Съществуващи решения

Синтезирането на глас е технология, която търпи постоянно развитие и подобрение. Системите, които я използват, се променят и разширяват, като целта е да се наподоби възможно най-много естествения говор, с което да се пригодят машините да изпълняват човешки функции. Ще разгледаме някои от съществуващите системи, които използват един или друг модел за решение на описаните проблеми.

2.3.1. *Google Translate* [1]

Една от най-популярните и лесни за използване системи, предоставяща трансформиране на текст до звук, е преводача на *Google*. Той поддържа голям брой езици, които имат почти перфектно произношение. Изглежда по начина, показан на Изображение 1. Проблема тук е, че не можем да свалим генерирания прочит. Това е частично преодоляно с предоставянето на потребителя достъп до синтезатора на глас чрез уеб услуги. Това предоставя част от желаната функционалност, но потребителя трябва да си напише програма, която да използва тези уеб услуги. Също така трябва да има постоянна Интернет връзка, което налага определени ограничения. Също така тази система не предоставя синтез за български език. Като цяло предоставяните услуги от *Google Translater* се от високо качество, но са сравнително ограничени и се нуждаят от разширяване.

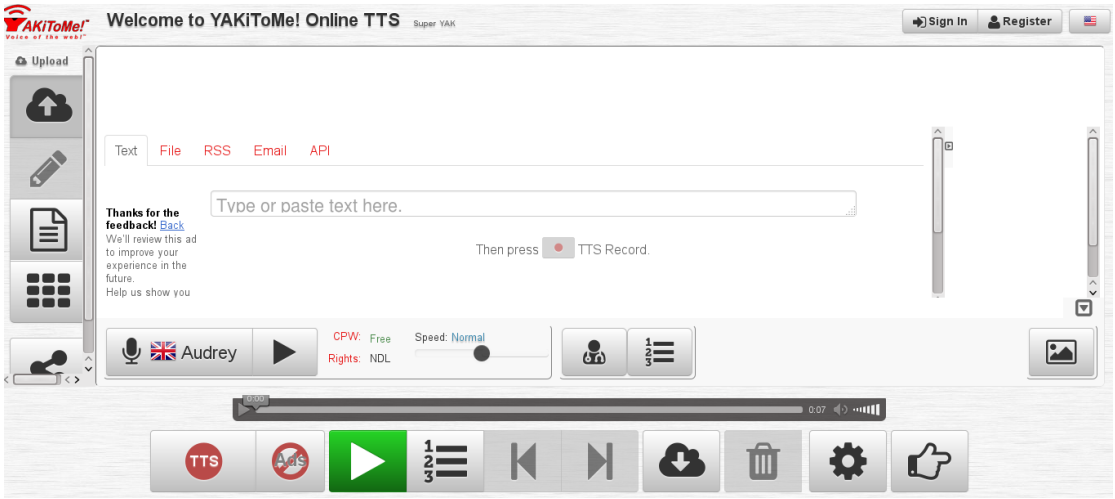


Изображение 1: Изглед на преводача на Google

2.3.2 *YakiToMe!*[2]

YAKiToMe! е безплатна онлайн система за прочит на текст, изглеждаща по начина, показан на Изображение 2. Тя е изцяло безплатна за всички регистрирани потребители. След регистрация се позволява качване на документи (*PDF*, *TXT*, *DOC*) и системата конвертира съдържанието им до аудио в mp3 формат. След което може да се свали генерирания файл на локалния компютър. Освен файлове, системата позволява трансформиране от емайли, *RSS feeds*, книги. Поддържат се голям брой езици, включващи Английски, Испански, Немски, Френски, Руски, Португалски и други. Не се поддържа български, което е сериозен недостатък. На потребителите е предоставен достъп до *Restfull API*, с

което може да се използват ресурсите на системата. Предлагат се конфигуриране на говора, което е добре при такъв тип системи, тъй като всеки потребител може да желае различна скорост, интонация, глас за четене.



Изображение 2: Изглед на TTS системата YAKIToMe!

Като цяло системата изглежда доста прилично и предоставя богат набор от функционалности. Но като начало, ако използваш безплатната версия постоянно се появяват различни ограничения, има дразнещи реклами и като всяка програма, която предлага платен и безплатен вариант, по всякакъв начин се прави опит да се накара потребителя да заплати определена сума. Освен това не се предлага възможност за инсталиране на системата на локалния компютър, за лично ползване, което налага ограничения на начина на ползване. Също така, не е възможно добавяне на функционалности от страна на потребителя. Системата е добра отправна точка, която след определени модификации би могла да се превърне в доста полезен продукт.

2.3.3 *iSpeech*[3]

iSpeech е платформа, предоставяща много възможности на своите потребители, изглеждаща по начина, показан на Изображение 3. Броя на приложенията, които *iSpeech* създава постоянно расте, като сферите на приложение са разностранни – обучение, шофиране, имитация на гласове и др. Платформата им за конвертиране на текст към звук, предлага определени настройки на генерирания глас – скорост на четене, тембър.

В сайта на системата са описани основните предимства на тази платформа:

- Първият *TextToSpeech* (*TTS*) продукт използващ облак (*SaaS*), очакващ патент
- Най-бързият, качествен и мощен *TTS* правен някога
- Възможно най-натурално звучащите гласове на пазара
- Без необходимост от инсталация за използване на *TTS*

- Безплатно използване на програмния интерфейс за тестването
- Огромен брой приложения за телефони
- Много поддържани файлови формати: *wav*, *mp3*, *ogg*, *wma*, *aiff*, *alaw*, *ulaw*, *vox*, *mp4*

Проблемът е, че те са достъпни само за регистрирани потребители, заплатили определена такса за ползване. Безплатните приложения имат ограничение на броя въведени думи. Гласовете не поддържат Български, като най-добре четящия глас на кирилица е руския, но при четене има руски акцент.

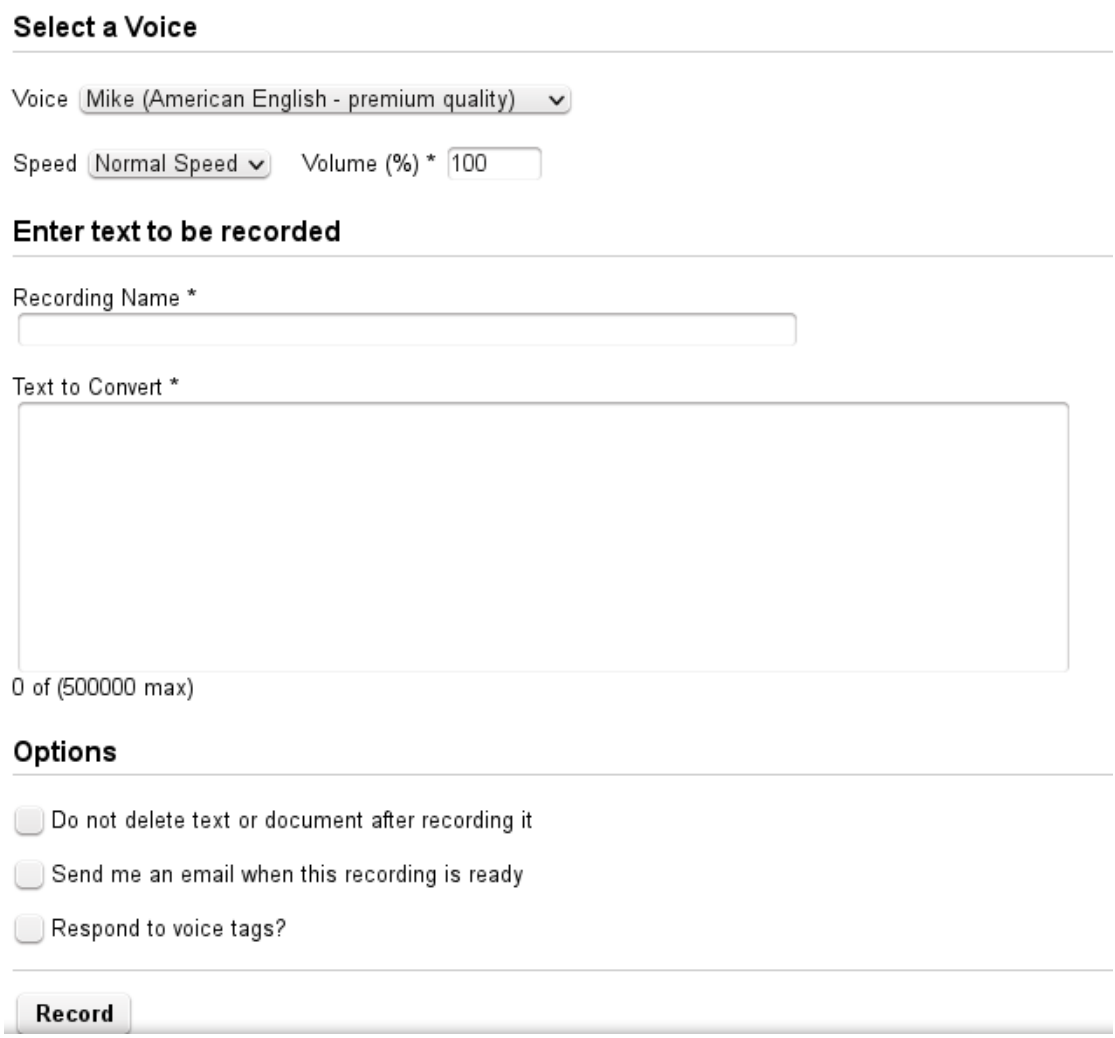
Изображение 3: Изглед на TTS системата iSpeech

2.3.4 *SpokenText*[4]

SpokenText е популярна и надеждна платформа за прочит на файлове, изглеждаща по начина, показан на Изображение 4. Използването му става със създаване на безплатен акаунт, валиден 7 дни, с който може да се тестват функционалностите на системата, но не е пригоден за дълготрайно използване, тъй като има ограничение на размера на качения файл, записа съдържа рекламни материали в началото и в края, броя на записите е 5 на ден.

Като цяло след заплащане функционалностите изглеждат сравнително прилични. Поддържаните файлови формати са – *Microsoft Word* и *PowerPoint*, текстови формати(*txt*), уеб страници и новини. Поддържа се конвертиране на няколко файла едновременно, но все пак има ограничение на размера (20-40 mb). Поддържаните езици са Английски, Немски, Френски и Испански, с разновидности по пол и акцент.

Дизайна е интуитивен и лесен за използване. Възможностите за разширение са почти никакви и не се забелязва друг начин за използване, освен чрез уеб браузър, което налага драстични ограничения за допълнителни разработки.



Изображение 4: Изглед на TTS системата SpokenText

2.4. Избор на критерии за сравнение и сравнителен анализ

От описаните методи, използвани за решаване на конкретна проблемна област, ще видим какво може да бъде използвано в *Ultimate Speaker*, като така ще улесним последващия анализ и имплементация.

Поради ограниченото време и насоката на нашия продукт, възможността за собственоръчно писане на каквито и да било модули отпада, защото ще отнеме изключително много време. За това трябва да изследваме внимателно съществуващите решения и библиотеки, да видим какво от тях може да бъде използвано и да изберем най-доброто. В следващата глава ще направи детайлен анализ на съществуващите синтезатори на глас, библиотеки за извличане на информация, методи за предоставяне на отдалечен достъп до операциите на системата, възможностите за търсене и т.н. Основната ни цел е те да са безплатни, платформено независими, да предлагат възможност за разширяване (евентуално да са с отворен код) и да предлагат възможност за инсталация в дистрибутиран вид.

Сега ще направим сравнителен анализ на разгледаните в предната точка системи, които предоставят желаните от нас функционалности. Ще ги сравним, посредством таблица критерии/оценка, като оценката ще е число от едно до десет - Таблица 1. Така в последствие по-лесно ще можем да си направим изводите, и да преценим какво трябва да се променим в нашата система, за да може тя да е конкурентоспособна и да избегне проблемите и грешките, допуснати при вече съществуващите системи.

Критерии за оценка	***Google Translate***	***YakiToMe!***	***iSpeech***	***SpokenText***
Отдалечен достъп	8	10	10	0
Offline използване	0	0	8	0
Отворен код	3	2	2	0
Безплатно ползване	10	3	3	5
Възможности за разширяване	0	0	0	0
Търсене в файловете	0	0	0	0
Интуитивен интерфейс	10	9	6	10
Качество на генерирания глас	10	10	10	10

Таблица 1: Сравнителен анализ на готовите решения

2.5. Изводи

Както виждаме, разгледаните системи имат няколко проблемни точки, в които нашата система може да ги изпревари значително. Тя задължително трябва да предоставя възможност за отдалечен достъп, като за това да се използват най-новите технологии, с цел задоволяване на всички потребности на потребителя. Системата ще позволява достъп от уеб браузър и възможност за локално инсталиране, като така ще удовлетворим потребителите със слаба Интернет свързаност. *Ultimate Speaker* ще е безплатна, с отворен код, а от там и лесна за разширяване, като така ще привлече по-голям брой като обикновени потребители така и разработчици, желаещи да модифицират системата за своите нужди. *Ultimate Speaker* ще предоставя възможност за търсене в качените документи, което не е възможно в нито едно от съществуващите решения. Интерфейсът трябва да е възможно най-лесен за използване и интуитивен, като така целим да не натоварваме потребителя и да го задържим възможно най-дълго. Единственото място, в което можем да отстъпим по качество, е това на генерирания глас, тъй като предлаганите безплатни синтезатори не са от най-високо качество.

Глава 3. Използвани технологии

В тази глава подробно ще разгледаме всички използвани средства (технологии и платформи) при създаването на *Ultimate Speaker*. Ще започнем с основните изисквания към тях, ще анализираме и сравним възможните варианти, ще изберем какво да използваме и накрая ще направим извод.

3.1. Изисквания

Използваните технологии трябва да отговарят на няколко основни изисквания, за да бъдат класифицирани като подходящи за използване в нашия продукт:

- Платформено независими
- Безплатни и с отворен код
- Възможност за разширение
- Да могат да се използват в дистрибутирана среда

Оценявайки всеки един инструмент, по тези основни характеристики и други, специфични за проблема който решава, ще направим класификация и ще изберем най-подходящия, който в последствие ще използваме.

3.2. Видове технологии, начин и място за използването им - сравнителен анализ

3.2.1 Избор на синтезатор за глас.

Развитието на двете основни технологии свързани с гласова интерпретация - *Text-to-speech* и *Speech-to-text* доведе до излизането на пазара на множество продукти, предлагащи различна имплементация. Инструментите, които привлякоха вниманието при направата на проучването, са следните:

- **Синтезатор *SpeechLab* („Гергана“)**[5]

  Продуктът ***SpeechLab***, с неговия глас „Гергана“, е първият висококачествен синтезатор на българска реч. Въпреки че към момента има още две алтернативи, гласа „Гергана“ си остава най-предпочитания от масата незрящи, особено за работа (поради добрия баланс между пъргавост и отчетливост). Като негов недостатък може да се изтъкне липсата на поддръжка на по-рядко използвани знаци от *Unicode* кодовата таблица, като например знакът за неравенство (≠), и др. подобни. Освен с четенето на текстове на български, *SpeechLab* се справя прилично добре и с четенето на текстове на английски, което е предимство.

  SpeechLab се разпространява безплатно, но само за хората със зрителни увреждания в България. Хората със зрителни увреждания получават безплатен индивидуален лиценз за некомерсиално ползване на *SpeechLab* 2.0 от Фондация "Хоризонти" или от Съюза на слепите в България.

  Инсталирането на ***SpeechLab*** е много лесно. Стартира се изпълнимия

файл на инсталиращата програма, щрака се няколко пъти на бутона ***Next***, след това на бутона ***Install***, а накрая – на бутона ***Finish*** и инсталирането е завършено. Препоръчително е след края на инсталацията да рестартирате компютъра. След това гласа „Гергана“ („*Gergana*“) ще бъде наличен за избор в екрана за настройка на текст-към-реч в контролния панел на *Windows*, както и в настройките за *SAPI5* в съответния екранен четец.

- **Синтезатор *Innoetics Irina* („Ирина“)**[6]

Гласът „Ирина“ е разработен от гръцката фирма „*Innoetics*“. Негово основно предимство е естественото звучене на гласа. Платен е (към текущия момент – 39 евро за един самостоятелен лиценз); макар и достатъчно отчетлив и приятен за слушане на текст на български, гласа „Ирина“ не се справя толкова добре с отчетливостта при четене на текстове на английски.

- **Синтезатор *eSpeak* (и с български глас)**[7]

Многоезичният речев синтезатор ***eSpeak*** е едно напълно безплатно и с отворен код решение за прочитане на текстове на глас на множество езици. Негов главен недостатък е в качеството на изходната реч – повечето потребители (особено тези свикнали с по-висококачествените речеви синтезатори), намират качеството на ***eSpeak*** за незадоволително. Едно от предимствата му е в пъргавостта (времето за реакция) при работа. Това обаче е за сметка на отчетливостта (разбираемостта) му. Друго негово предимство е това, че може коректно (макар и в повечето случаи – на английски) да произнася доста по-голям набор от знаци от ***Unicode*** кодовата таблица, в сравнение със ***SpeechLab***.

Сдобиването с ***eSpeak*** става чрез изтегляне на инсталационния му пакет от официалния му уеб сайт: http://espeak.sourceforge.net, от раздела „***Download***“.

3.2.2 Избор на инструмент за извличане на текст от различни файлове

Извличане на текст от файл е сравнително популярна операция и е добре да се разгледа възможността да се използва готов инструмент, позволяващ желаните трансформации. Дори да не поддържа всички възможни файлови формати, ще е нужно да пишем алгоритъм единствено за липсващите такива, което значително ще улесни разработването на системата. Съществуват няколко готови библиотеки, предоставящи желаните функционалности.

Apache POI[9] е инструмент позволяващ манипулация на широк спектър от файлови формати, основно използвани в *Windows* операционните системи. Хубавата част е, че е разработен специално за *Java*, което улеснява използването му в конкретната система. Също така ***Apache POI*** е специално пригоден за работа с файлове от офис пакета на *Microsoft* (*Excel*, *Word*, *PowePoint*, *Publisher* и други). Но след преглед на документацията, се установи, че самите разработчици препоръчват използването на ***Apache Tika***, за професионално извличане на информация, тъй като тя предлага и много други полезни функции, надхвърлящи многократно тези на ***Apache POI***.

Apache Tika[10] е инструмент със свободен лиценз, позволяващ анализ и извличане съдържанието и мета информацията на огромен набор от файлове,

включително популярните *PDF* и *PPT*. Този инструмент използва и разширява възможностите на ***Apache POI,*** като добавя допълнителна функционалност за разпознаване на език, съдържание, тип на файла и много други. Броя на файловите формати, поддържани от този инструмент, е огромен, което ще направи разширяването на ***Ultimate Speaker*** изключително лесно. Има възможност за дефиниране и добавяне на собствени формати и анализатори, което на практика е всичко от което се нуждаем.

3.2.3 Предоставяне достъп на крайния потребител.

За да бъде възможно най-гъвкава и удобна за използване, системата трябва да позволява:

- Инсталиране на клиентска машина, за офлайн достъп
- Инсталиране на системата на публичен сървър и позволяване на публичен достъп през уеб браузър.
- Достъп до системата чрез уеб услуги

За да удовлетворим и трите възможни варианта, системата трябва да предоставя два отделни компонента – клиент и сървър. Сървърът ще предоставя достъп до функционалностите си чрез уеб услуги, а клиента ще използва тези услуги за да визуализира състоянието на системата. При тази конфигурация ще имаме следните решения:

1. При инсталиране на клиентската машина, ще се слага както клиентската, така и сървърната част. Така клиента ще може да избира дали да се свърже към публичен сървър или към локален, като локалния ще вдига уеб услугите на локален адрес, от където клиента ще може да ги достъпи, дори при ограничен интернет достъп.
2. На публичния сървър ще бъде инсталиран както клиента така и сървъра, с цел да се предостави достъп на клиентите както през уеб браузъра (без да инсталират нищо) така и през клиентския софтуер, който да се връзва към уеб услугите.
3. Сървърната част ще предоставя достъп чрез уеб услугите, като тока ще отпаднат всякакви ограничения откъм клиентската част.

Тъй като *Restfull* уеб услугите са най-популярен, използван и интуитивен метод за предоставяне на отдалечен достъп, те ще бъдат използвани при създаването на *Ultimate Speaker*. Целта е да се избегне обвързване с остарели технологии, които затрудняват разработката. Комуникацията между бизнес логиката и потребителския интерфейс ще става с помощта на ***Java JAX-RS*** имплементация. Ще разгледаме някои от най-популярните платформи, предлагащи бързо и лесно разработване на уеб услуги.

Jersey[11] е широко използвана имплементация на ***JAX-RS***, създадена от *SUN Microsystems* в последствие поддържана от *Oracle,* която разширява допълнително спецификациите с цел по-лесното разработване на **Restfull** клиент и сървър. Позволява допълнително разширяване, в зависимост от потребностите на всеки потребител. Включва поддръжка на *Oauth, Atom, MVC, FastInfoset*.

Предлага възможност за интеграция с *Apache HTTP Client, Guice, Spring, MOXy* и много други.

RestEasy[12] е проект на *Jboss* и е изцяло сертифицирана имплементация на ***JAX-RS*** спецификацията. Може да работи със всеки един уеб сървър (*Jboss, Tomcat* и др.). Има внедрена имплементация за тестване на създадената ***Restfull*** система. Предоставя допълнителна възможност за лесно писане на клиентска част, която да си комуникира със създадената уеб услуга. Използва различни видове сървърна кеш с цел бързодействие. Има богат набор от трансформатори за *XML, JSON, YAML, Fastinfoset, Multipart, XOP, Atom* и др. Използва *JAXB* сериализация на обекти в *XML, JSON, Jackson, Fastinfoset* и *Atom*, като също така има абстракции за основните колекции съдържащи *JAXB* обекти. Предлага автоматично компресиране/декомпресиране в клиентската и сървърната част, използвайки *GZIP.* Предоставя асинхронна абстракция на *HTTP(Comet)* за *Jboss* и *Tomcat.* Използва асинхронно изпълнение на задачите. Много богат, лесен и удобен начин за прихващане на заявките. Поддържа криптиране и дигитален подпис с помощта на *S/MIME* и *DOSETA.* Включва още много други възможности като например *EJB, Seam, Guice, Spring* и *Spring MVC.*

Restlet[13] е платформа, подпомагаща разработването на програмни интерфейси, които следват архитектурата на *REST*. Много мощен, но в същото време сравнително сложен инструмент. Възможност за разработване на *REST* приложения от ниско ниво. Позволява голяма част от описаните възможности на другите инструменти, като в основната имплементация е включена както сървърна така и клиентска част. Предимството тук е, че можем да използваме този инструмент без уеб сървър, което ни позволява да избегнем обвързването и настройването на конкретен сървър, което би опростило значително имплементацията. Също така *Restlet Framework*-а работи на голям набор от платформи - *Java SE/RE, OSGI Environment, Google App Engine, Google Web Toolkit, Android.*

При избор на технология за създаване на клиентската част от програмата, трябва да се вземе в предвид какво ще предоставя сървъра и какво точно се очаква от клиента. След като сървъра ще използва *Restfull* уеб услуги, единственото което клиента трябва да прави е да изпраща *Ajax* заявки и да показва статуса на системата. Това може лесно да бъде направено без използването на сложни уеб сървъри, които да пускат и спират клиентското приложение. Не е нужно нищо повече от чист *HTML*, *CSS* и *JavaScript.* Така клиентското приложение ще стане изключително опростено, а разширяването му много лесно. За улеснение на имплементацията могат да бъдат използвани различни *JavaScript* и *CSS* библиотеки и скриптове, като така голяма част от логиката остава скрита за крайния потребител. Възможни библиотеки, подходящи за разработваната система включват:

1. ***Bootstrap***[14] – използван за лесно подравняване на страници, без използване на таблици, позволяващи динамично оразмеряване
2. ***Jquery***[15] – библиотека улесняваща и разширяваща използването на голям брой JavaScript функции. Към самата библиотека има множество плъгини, които добавят някаква функционалност:

- ***JQuery UI***[16] – комбинация от множество визуални компоненти, използвани при комуникация с потребителя, ефекти, теми и др.
- ***JQuery Form***[17] – това е плъгин, който позволява HTML формите да използват AJAX, което може да бъде полезно при имплементиране качването на файлове.
- ***JQuery Raty***[18] – това е плъгин, улесняващ създаването на рейтинги. Той може да бъде използван при разработването на рейтинги за качените файлове.
- ***JQuery UploadFile***[19] – плъгин, който позволява качването на файлове чрез избиране или ***Drag&Drop***.

3. ***Reveal***[20] – това е библиотека за показване на онлайн презентации. Съществуват много подобни решения – ***impress.js***[21]***, www-lectures***[22]***, deck.js***[23]***, html-slideshow***[24] и т.н.,

Избора тук не е от голямо значение, тъй като на потребителя се предоставя целия код на приложението и ако нещо не се хареса или предпочитанията са към друга библиотека за конкретна операция, клиента може да промени съществуващата имплементация.

3.2.4 Избор на система за съхранение на данни.

Всяка система има нужда от база от данни. Това е сигурно място, където програмата може да запазва всякаква информация, с цел по-лесен и бърз достъп, по-оптимизирано откъм размер на файловата система, възможност за дистрибутиране на информацията на няколко машини и т.н. Съществуват много варианти на бази от данни – платени, безплатни, локални, външни и т.н. Като цяло всеки един производител се стреми да предостави възможно най-доброто на своите клиенти, и основните разлики са в броя на поддържаните потребители и поддържания обем на съхранявана информация. Базата е модул, който може да бъде заменен лесно и за това избора ще се ориентира към по-познат продукт, от колкото към най-качествен, с цел улесняване на разработката. За да премахне голяма част от изискванията към базата от данни, ще разделим информацията, като така ще минимизираме размера на запазваните данни. Системата основно ще се съхранява същинския файл, от който ще извличаме съдържание, самата извлечена информация и генерирания прочит. Това са статични данни, които няма да се променят за конкретен файл. За това тази информация ще се съхраняват директно на файловата система, с цел намаляване размера на базата от данни. В самата база ще се пазят единствено пътища до съответните файлове. Именно по тази причина ще се ориентираме към база от данни, която е възможно най-лека и бърза, в която да могат лесно да се съхраняват сравнително малки по обем данни.

3.2.5 Избор на технология за търсене.

Предоставяне на потребителите възможност за търсене е основно предимство, което нашата система ще има пред останалите подобни решения. Търсенето е изключително полезно за бързото намиране на конкретен файл, ако имаме множество такива или просто сме забравили къде какво се съдържа.

Търсенето стана неизменна част от новите системи и потребителите са свикнали да го използват до такава степен, че като срещнат система предоставяща огромно количество информация, но без възможност за търсене, са пренасочват към друга подобна платформа.

Като знаем основните критерии за сравнения и разгледаме популярните библиотеки предоставящи тази функционалност, можем да изберем измежду следните готови решения (всичките са базирани на *Java* с цел по-лесното интегриране):

1. ***Apache Lucene***[27] – това е ядро, предоставящо основните функционалности по индексиране и търсене във файловете. Сравнително стар и стабилен проект на *Apache*, но не особено лесен за използване, тъй като предоставя само основната функционалност и трябва да се дописват довършителните неща. Именно поради тази причина *Apache* разработва следващите два продукта, които усъвършенстват този.
2. ***Apache SOLR Search***[28] – изключително бърза и удобна за ползване платформа, предоставяща всички възможности на *Apache Lucene*, надграждайки ги с цел улесняване на използването им. Платформата е надеждна, скалируема, позволяваща дистрибутиране на индексите, репликация, разпределяне на натоварването между отделните системи, автоматично възстановяване при грешки и много други. Това е платформата, използвана от множество уеб сайтове по света. Единственият и недостатък е, че дистрибутирането става малко по-сложно, и особено в по-старите версии не е особено надеждно.
3. ***Apache Eleastic Search***[29] – най-новата платформа за търсене на *Apache*, използваща отново Apache Lucene, но изградена на различен принцип от *Apache SOLR*. Тъй като *SOLR Search* е стара платформа и поддържа изключително много стари неща, поддържането и в крак с новостите е проблем. Именно за това се появява и *Elastic Search* – изграден като дистрибутиран индекс от самото начало, интегриращ се лесно посредством *RestAPI* заявки, позволяващ анализ в реално време, поддържащ много мощно търсен в целия текст на документа (*full-text search*) и много други.

3.3. Избор на средствата

Синтезатор на глас - наличието на Български език в ***eSpeak***[8] и възможността за използването му без заплащане на лиценз, наклонява везните в негова посока. Архитектурата на системата трябва да позволява замяна на избрания инструмент, като единственото което трябва да се направи е да се добави имплементация за нов синтезатор на глас и да се смени инициализирането на синтезатора при старт на системата.

Библиотека за извличане на информацията - след като в документацията на ***Apache POI*** се препоръчва използването на ***Apache Tika*** за по-професионално решаване на проблема с извличане на информация от текст и наличието на предишен опит в работата с този продукт, предопределя избора. ***Ultimate Speaker*** ще използва ***Apache Tika*** за извличане на информацията от файловете.

При необходимост съществуващите имплементации могат да бъдат променени или разширени, с цел оптимизиране на работата им и адаптирането им към нуждите на система.

Технология за предоставяне отдалечен достъп до сървъра – тъй като вече е решено, че ще се използва *Restfull* уеб услуги и факта, че възможните решения са приблизително еднакво удобни и лесни за употреба, ще вземем субективно решение при този избор. Богатия опит в работата с *Restlet* имплементацията и удовлетвореността от възможностите които предоставя, предопределя избора и за разработване на *Restfull* частта на *Ultimate Speaker*.

Клиентска част на приложението – както споменахме вече, клиентската част е направена за да покаже възможностите на приложението и няма задължителен характер. Потребителя има свобода да разработи свое собствено клиентско приложение, за това използваните тук технологии не са от голямо значение. Единствено се ограничаваме до използването на *HTML*, *CSS* и *JavaScript*, с цел избягване на допълнително обвързване с уеб сървъри.

База от данни - Тя трябва да е достъпна възможно най-бързо и с цел оптимизиране на търсенето и сортирането на всички файлове, ще се спрем на ***SQLite***[25]. Избраната имплементация е на ***Xerial***[26]**,** защото включва компилирана версия на *SQLite* за основните платформи (*Windows, Mac OS, Linux)*, както и *JDBC* драйвер за връзка към базата, в един единствен архив. Така се избягва обвързаността с определена платформа, като в същото време използването на базата е изключително опростено, понеже всичко се съхранява просто в един файл на файловата система и няма отделен сървър, който трябва да се пуска преди да можеш да работиш с базата. Друго добро качество на *SQLite* е, че може да използва изключително малко памет, което го прави добро решение за платформи с ограничени ресурси – като таблети, телефони и много други.

Платформа за търсене – от разгледаните платформи търсим най-новата и бърза, с цел максималното удовлетворим клиентите. Известно е, че голяма част от фирмите и сайтовете, използващи *Apache SOLR Search* започват да използват *Elasticsearch*, което накланя везните при избора за система за търсене. Също така наличието на предишен опит в работата с *Elasticsearch* и с цел улесняване на имплементация, ще се спрем на нея при разработване на модула за търсене.

3.4. Изводи

При избора на технологии и средства се съобразяваме с основните критерии за оценка и в последствие избираме най-добрия, най-новия или най-познатия от възможните инструменти. Така гарантираме, че нашата система ще е в крак с новостите, много по-лесно ще бъде конфигурирана за конкретните потребителски нужди и ще задоволи на максимум изискванията на клиентите – за производителност, лекота на използване и удобство.

Глава 4. Анализ

Анализа е съществена част от създаването на системата. Той има ключова роля за последващото удовлетворяване на потребителските изисквания. Тук трябва да се определи от какво се нуждае система и да се вземе в предвид при проектирането.

4.1. Концептуален модел

Системата трябва да е разделена на два основни модула – клиентски и сървърен. Тази архитектура е изключително популярна и може да удовлетвори различни потребителски изисквания. Клиентската част трябва да е възможно най-олекотена, за да можем да предоставим възможност за комуникация на клиенти с ограничени ресурсни възможности (включително таблети, телефони). Сървърната част ще се грижи за комуникацията с клиентите и обработването на техните заявки. При желание, потребителя може да инсталира и двата компонента на един и същ компютър и така да притежава пълното приложение, без да се налага да осигурява постоянен достъп до сървъра.

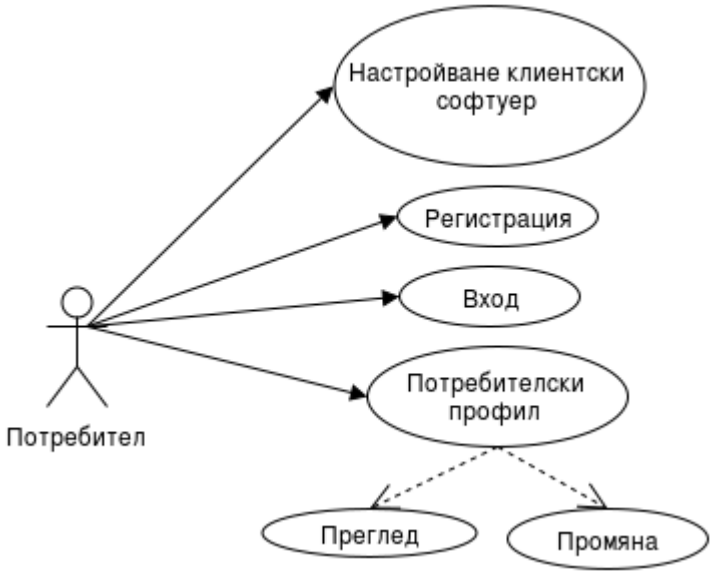
Комуникацията между клиента и сървъра ще се извършва с *Restfull* заявки. Така се премахва обвързаността с конкретна клиентска имплементация и се дава възможност на потребителите да разработят свое приложение или да разширят вече съществуващо такова, да използва и нашата система. Също така промените в сървърната част могат да останат невидими за клиентите, с което правим поддръжката много по-лесно осъществима.

4.2. Потребителски (функционални) изисквания

Всеки един потребител на системата, трябва да може да изпълнява серия от действия, някои от които свързани по между си. Може да се обмисли създаване на потребителски групи/роли, като всяка една група да има определени права – най-простите такива групи са администратори и потребители. При евентуално такова разделени, изискванията за различните групи ще са различни. За момента системата трябва да поддържа само едната от тези групи – потребители, за това сега ще опишем изискванията за нея, а при нужда, ще разработим модел за нови групи от потребители. При платените системи има разделение по функционалности, като колкото повече права иска един клиент, толкова по-голяма сума трябва да заплати, но нашата система е безплатна и такова разделение няма да има.

Ще разгледаме основните изисквания, които системата трябва да удовлетвори, за групата на обикновените потребители. Като за начало трябва да се определи кой може да стане потребител на системата. Тъй като тя е безплатна, то всеки един клиент имащ достъп до сървъра, трябва да може да използва възможностите му. Тук идва и първото и основно изискване, което трябва да бъде изпълнено – регистрация на потребител. След като се регистрира, потребителя трябва да може да влезе в системата и да започне да я използва. Преди идентифициране на потребителя, системата трябва да предоставя само ограничени функционалности, като регистрация, вход и евентуално настройка на клиентската част от програмата (примерно

комуникация със сървъра). След идентификация, потребителя трябва да има пълен достъп до предлаганите от системата операции, включително и промяна на индивидуални настройки. Процеса по регистрация/вход е визуализиран в потребителския случа, показан на Графика 1. Последователността е от горе на долу, което значи че не можем да имаме вход, преди регистрация, регистрация преди да сме настроили връзката със сървъра и т.н.



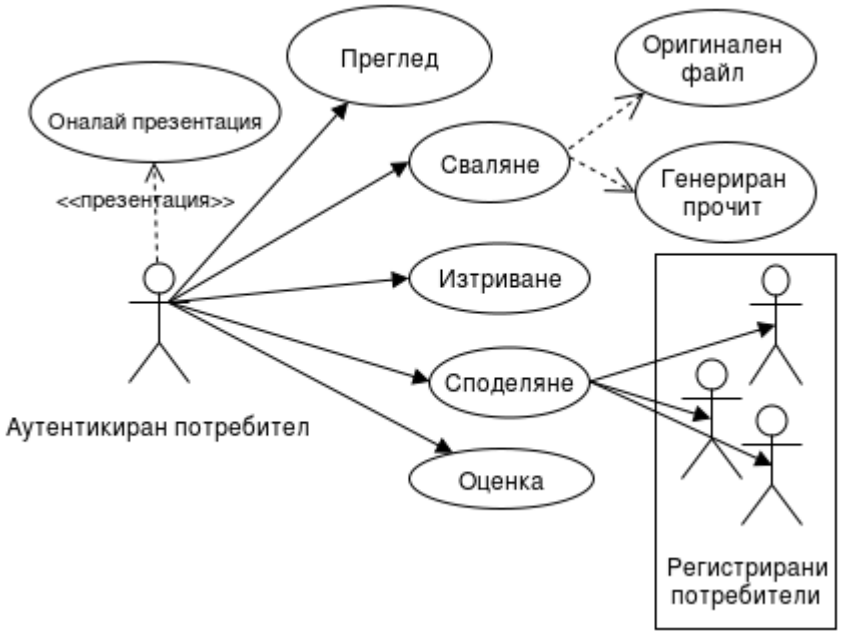
Графика 1: Потребителски случай - регистрация/вход в системата

След като сървърната част е разпознала потребителя, той трябва да има достъп до функционалностите на програмата, които могат да бъдат разграничени в три основни групи – качване на файлове, преглеждане на всички файлове, до които имаме достъп и търсене. Ще разгледаме всяка една група и ще опишем накратко какво ще очаква потребителя от нея.

Качването на файл е сравнително проста операция, от гледна точка на клиента. Потребителя трябва да има поне две възможности за качване на конкретен файл/файлове – избор от локалния диск и „*Drag&Drop*“, тъй като те са доста популярни и се идентифицират като нормален начин за предоставяне на информация. Възможно е също така да се предостави на потребителя поле , в което да въведе свободен текст, който да бъде качен в системата като файл. Тази операция няма да бъде илюстрирана като потребителски случай, тъй като включва само едно действие от страна на потребителя - предоставяне файл на системата.

Преглеждането на файлове трябва да позволява на потребителя лесно и удобно преглеждане на всички свои файлове, като под свои можем да разбираме качени лично от него и евентуално споделени от други потребители. Визуализацията трябва да позволява на потребителя да предприеме определен набор от действия, върху файловете които вижда – сваляне, споделяне, изтриване, евентуално преглеждане на съдържанието в онлайн презентатор,

добавяне на рейтинги и бележки. Някои от операциите са зависими от типа на файла, тъй като онлайн презентатора трябва да е възможен само за презентационни файлове. Преглеждането на файлове е визуализирано в потребителския случай, показан на Графика 2. Както се вижда, споделянето също е зависима операция, защото ако няма други регистрирани потребители в системата, няма да има с кого да споделим файла.



Графика 2: Потребителски случай - преглед на файлове

Третата група потребителски функционалности е търсенето. Това е жизнено важна операция за системата, тъй като тя дава огромно предимство пред останалите системи с подобно предназначение. Потребителя трябва да може да търси по имена на файлове или по тяхното съдържание. Системата трябва да поддържа търсене на фрази и евентуално търсене с маска (*wildcard*). Трябва да се направи видимо за потребителя къде са били съвпаденията на търсените термини, с цел бързо определяне дали даден резултат е този, който потребителя търси. Резултатите от търсенето трябва да позволяват на потребителя сваляне на намерения файл.

Това са основните функционалности, които системата трябва да реализира и да предостави по удобен и интуитивен начин на потребителя. Сега ще обърнем малко по-голямо внимание на нефункционалните изисквания към продукта.

4.3. Качествени (нефункционални) изисквания

Това са изисквания, които системата трябва да изпълни с цел, както да удовлетвори потребителите, така и да има предимство пред останалите подобни системи.

Ще започнем с изискването за преносимост на приложението. То трябва да може да работи на всички популярни операционни системи – *Windows* и *Unix*. За

да се изпълни това условие, средствата използвани за разработка трябва да са системно независими. За това системата ще бъде разработвана основно използвайки езика за програмиране *Java*. Трябва да се внимава по време на разработка, тъй като има доста разлики, между двата класа системи и трябва да се проведат тестове на поне една версия на *Windows* и една на *Unix*. При избора на готови библиотеки за използване, условието за преносимост е със задължителен характер. Премахвайки ограничението за работа на специфична платформа, продукта ще може да задоволи всички потребители, независимо от предпочитаната от тях операционна система.

Програмата трябва да позволява два начина на използване – онлайн и офлайн. Това значи, че потребителите трябва да могат да избират да инсталират само клиентското приложение и да се свържат с публично оповестен сървър, или да инсталират и двете части на програмата, като така да я използват без достъп до Интернет. Това изискване трябва да се удовлетвори при спазване на архитектурата – клиент-сървър.

Като всяко едно приложение от клиент-сървър тип, трябва сериозно да се помисли за сигурността на предаваната информация. Комуникацията между двете части на програмата трябва да е криптирана, като така ще се затрудни възможно най-много евентуалното подслушване на трафика и кражбата на важна информация. Може да се обмисли поддръжка на различни начина за вход в системата – с парола, със сертификат, аутентикация чрез *LDAP* сървър и т.н.

Приложението трябва да позволява разширение до възможност за инсталиране на сървърната част в дистрибутиран вид. Така програмата ще може да е хоризонтално скалируема – което значи, че при голямо натоварване, към съществуваща инсталация може да се добави нов компютър, който да поеме част от натоварването на съществуващите сървъри. За тази цел използваните външни приложения трябва да поддържат използване в дистрибутирани системи. За целта на дипломната работа, системата ще работи само на една машина, но подготовката и за евентуално разширяване, трябва да се вземе в предвид по време на проектирането и разработката.

Системата трябва да е разширима и безплатна за използване. За тази цел кода на приложението ще е отворен и публично достъпен. Всеки който желае, може да го свали и промени така, че да отговаря на индивидуалните му изисквания. Удовлетворявайки това изискване, тази система ще стане една от малкото подобни, които предлагат такава свобода на своите потребители.

Голяма част от системите, предоставящи подобни функционалности са уеб базирани и с доста модерен и удобен интерфейс. За да може да се конкурира с тях, клиентската част на приложението трябва да е лесна за използване, интуитивна и изчистена. За да може да се използва както на обикновен компютър, така и на телефон/таблет – клиентското приложение трябва ще е уеб базирано, разработено използвайки най-новите технологии, с цел работа на различни уеб браузъри и иновативна визия.

Друга важна характеристика, която трябва да се опита да удовлетвори нашето приложение, е качеството на генерирания глас. Тъй като това е програма, приспадаща към класа на *textToSpeech* системите, важна характеристика е колко

лесен за разбиране ще е генерирания глас. Важно е също така потребителя да има възможност да променя някои от характеристиките на този глас – като сила, интонация, тембър, скорост на четене и т.н. Това се налага, тъй като всеки клиент може да има различни предпочитания за гласа, с който иска да слуша прочитите на своите файлове.

Като последно изискване, но не на последно място, ще посочим качеството на кода на приложението. При разработка трябва да се спазват основните правила за качествен програмен код, да се прилагат шаблони за програмиране, да се пишат тестове на програмните единици и т.н. Всички тези неща ще спомогнат за улесняване поддръжката на приложението, което е неразделна част от жизнения цикъл на всяка една система.

4.5. Изводи

Резултатите от анализа на всеки софтуерен продукт, се използват при нейното проектиране. За да улесним процеса на проектиране, ще представим обобщението от анализа във вид на таблица, съдържаща по един ред за всеки вид изискване (функционално/нефункционално). Ако този ред е удебелен, то това е задължително изискване, в противен случай то е с пожелателен характер. Така ще знаем какво трябва задължително да се включи при разработката, и кое може да се обмисли допълнително. Резултатите от анализа са представени в Таблица 2.

Функционални изисквания
Рзделение потребителите на групи
Рекистрация/Вход в системата
Настройване на сървър/клиент
Индивидуални настройки за потребителя
Качване на файлове - Избор от локален файл + *„Drag&Drop“*
Качване на файлове - Качване от текстово поле
Операции върху файлове - сваляне оригинал/прочит, изтриване, споделяне, онлайн преглед, оценка.
Операции върху файлове - добавяне на бележки към прочит
Търсене в свободен текст и фрази
Търсене с *Wildcard* заявки
Нефункционални изисквания
Системна независимост, Онлайн/офлайн използване, Сигурност на комуникацията, Отворен код, Разширимост, Безплатно приложение, Уеб базирана клиентска част, Качествен програмен код
Дистрибутираност, Високо качество на генерирания глас

Таблица 2: Резултати от анализа на системата

Глава 5. Проектиране

В тази глава ще се спрем на структурата на *Ultimate Speaker*, модулите които го изграждат, тяхната функция, как си комуникират. Ще разгледаме различните видове интерфейси, които системата предлага, каква стратегията използва за съхранение на данни и като цяло детайлно преглеждане на всеки един компонент, изграждащ програмата.

5.1. Обща архитектура

Системата е от вида клиент-сървър, като имплементацията използва архитектурния шаблон за програмиране Модел-Изглед-Контролер(*model, view, controller* или *MVC*). Изгледа представлява клиентската част на приложението, а модела и контролера се намират в сървърната част. Това е направено с цел лесната замяна на който и да е от трите компонента, като така е възможно най-лесно всеки потребител ще може да пригоди системата според личните си нужди.

5.1.1 *MVC*

Това е архитектурен шаблон за програмиране, който цели разделяна на интерфейса на приложението от бизнес логиката. Така двата модула стават независими, а контролера е връзката между тях.

Модел - бизнес логиката(ядрото) на приложението е реализирана посредством няколко модула, всеки от които се грижи за конкретна функционалност на системата – извличане на информацията от файловете, синтез на звук, индексиране на извлечената информация с цел търсене. Съществуващите модули могат да бъдат усъвършенствани и към тях може да се добавят нови, с цел разширяване възможностите на системата.

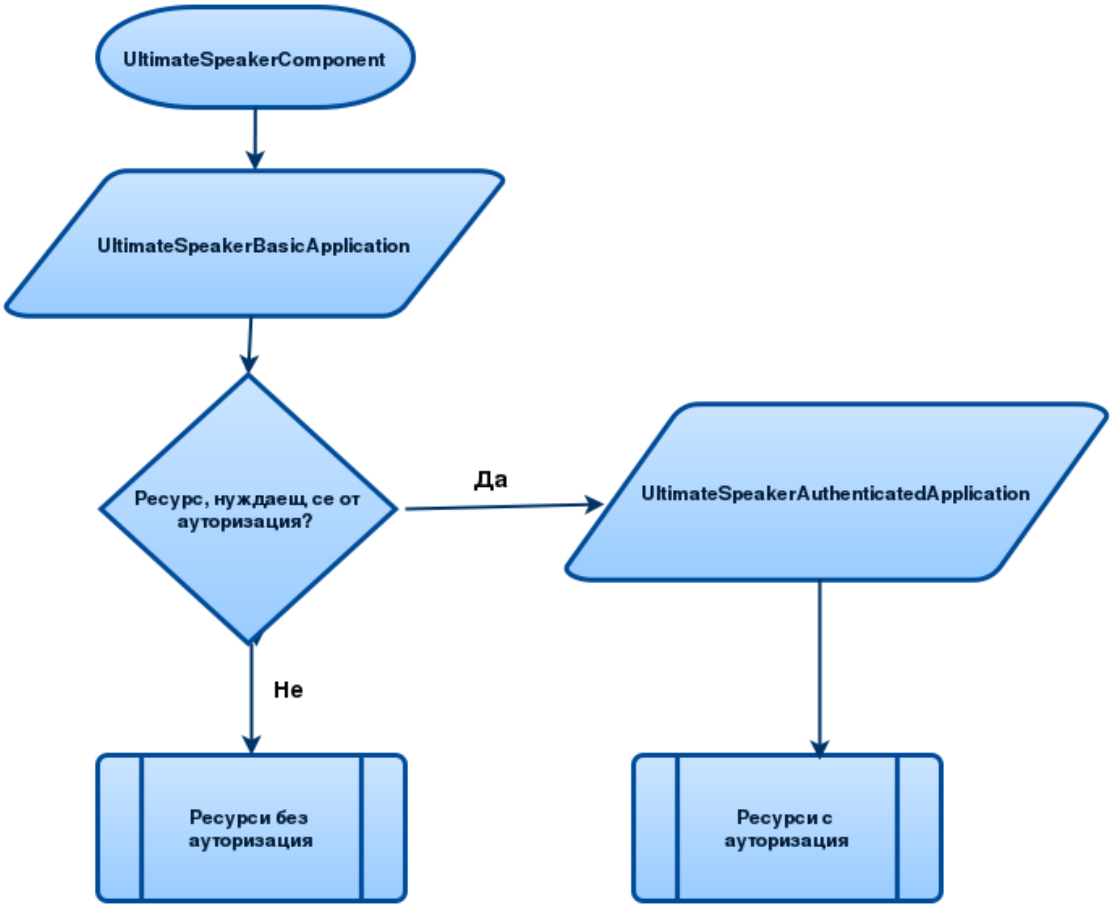
Изглед – представлява потребителския интерфейс, с който крайния потребител може да използва функционалностите на системата. Реализиран е възможно най-опростено, посредством *HTML*, *CSS* и *JavaScript*. Използвани са няколко съществуващи библиотеки на *JavaScript*, с цел улесняване на имплементацията. Тази част на приложението комуникира единствено с контролера, и няма представа каква точно е логиката на приложението. Това от което се интересува този модул е да покаже по подходящ начин информацията, до която даден потребител има достъп.

Контролер – това е частта от системата, която се грижи за осъществяване на комуникацията между модела и изгледа. Разработен е посредством *RestAPI* слоя, като така се премахват ограниченията към конкретен език за потребителския интерфейс. Работата на контролера е да предостави унифициран достъп на потребителя, като позволи достъп само до конкретни функционалности на модела, от които можем да се интересуваме. Конкретната реализация е описана в следващата точка.

5.1.1 *Rest API* слой

Това е контролера на системата. В предните точки и глави на дълго и на широко бе обяснено, защо е избрана точно *Restfull* технологията, за това тук ще

бъде разгледана конкретната имплементация. За реализирането на този слой се използва ***Restlet***[13] платформата, тъй като тя предоставя вграден *HTTP* сървър, много разширения, които могат да се включат при нужда и други. Наличието на предишен опит в интегрирането на платформата, допълнително накланя везните в нейна посока. Основната структура на този слой и избора на ресурс, който да обработи даден заявка, е показан на Графика 3.



Графика 3: Структура на Rest API слоя. Поток на събитията при избор на обработващ ресурс.

UltimateSpeakerComponent е стартовата точка на този слой. Класа е имплементация на *org.restlet.Component*. Тук са указани параметрите на *HTTP* сървъра, който системата ще използва. Сървърът ще използва порт, който се конфигурира от администраторския интерфейс, пускащ се при стартиране на приложението – подробности за него в точка 5.5. При стартиране на този компонент се стартира *HTTP* сървъра и може да започне обработката на заявки. Реалното обработване става в ресурсите, закачени за някое от приложенията. Всяко приложение указва *URL*-а, с който да достъпим конкретен ресурс.

UltimateSpeakerBasicApplication – това е базовото приложение, предоставящо достъп до ресурсите, които не се нуждаят от предварителен вход в системата. Това са потребителския ресурс (*UserResource)* и ресурса, предоставящ

документация (*DocumentationResource)*. Тези ресурси не се нуждаят от ауторизация, тъй като потребителския ресурс се използва за вход в системата, а посредством ресурса за документация, потребителя трябва да може да види какви заявки приема системата.

Потребителски ресурс(*UserResource)* – може да бъде достъпен посредством заявка до ***https://<IP>:<PORT>/users***. Поддържат се два метода:

- *POST* - заявка за регистрация на потребител, като в тялото на командата се изпращат нужните параметри – потребителско име, емайл и парола.
- *PUT* - заявка за вход на вече регистриран потребител.

Ресурс за документация *(DocumentationResource)* – това е ресурса предоставящ на потребителя описание на поддържаните заявки, тъй като едно *Restfull API* е безполезно, ако няма вариант клиента да разбере какво може да прави със системата. Документацията е във ***WADL***[30] формат, генерира се автоматично за всеки ресурс и е имплементирана посредством *WADL* разширението на *Restlet.* Самото разширение е подобрено, тъй като резултата, от генерираната по подразбиране документация, е незадоволителен. Цялата документация е достъпна в *HTML* формат на следния адрес: ***https://<IP>:<PORT>/api-docs*** и изглежда по начин, показан на Изображение 5.

UltimateSpeaker RESTful application

Receives RestAPI calls to operate with the system.

Summary

Resource	Method	Description
https://127.0.0.1:8181/users	POST	Description: Register user.
	PUT	Description: Authenticate user.
https://127.0.0.1:8181/api-docs	GET	Description: Generate information about all RestAPI resources.
https://127.0.0.1:8181 /management	GET	Description: UltimateSpeaker RESTful authenticated application - for detail information click the URL from example. Example: https://127.0.0.1:8181/management/?method=options

Resources

https://127.0.0.1:8181/users

Users resource: Provide operation over users.

Изображение 5: Изглед на онлайн документацията на RestAPI слоя

Както се вижда описанието започва с обобщение на всички *URL*-и, които се поддържат от системата, а в последствие има детайлно описание на всеки от тях, разделен по методи. Например за регистрация на потребител, имаме описанието, показано на Изображение 6. В началото имаме описание на самия метод, след това са параметрите които трябва да предоставим, дали те са задължителни или не, какъв е типа на заявката която очакваме (в този случай *application/x-www-form-urlencoded*) и накрая, в зависимост от статуса, какво можем да очакваме.

Избран е да се използва този метод за генериране на документация, тъй като е

удобен, голяма част от описанието се генерира автоматично, лесен е за разбиране от потребителя, а самия *WADL*[30] е популярен език за описание на уеб услуги.

Изображение 6: Описание на метод за вход на потребителя, от Потребителския ресурс

UltimateSpeakerAuthenticatedApplication – това е приложение, с което си комуникират регистрираните потребители. Всички ресурси от него използват базова аутентикация за верифициране, че даден потребител съществува в системата. Ресурсите които се поддържат, могат да се видят нагледно в документацията на приложението на адрес : ***https://<IP>:<PORT>/ management/?method=options*** , и са показани на Изображение 7.

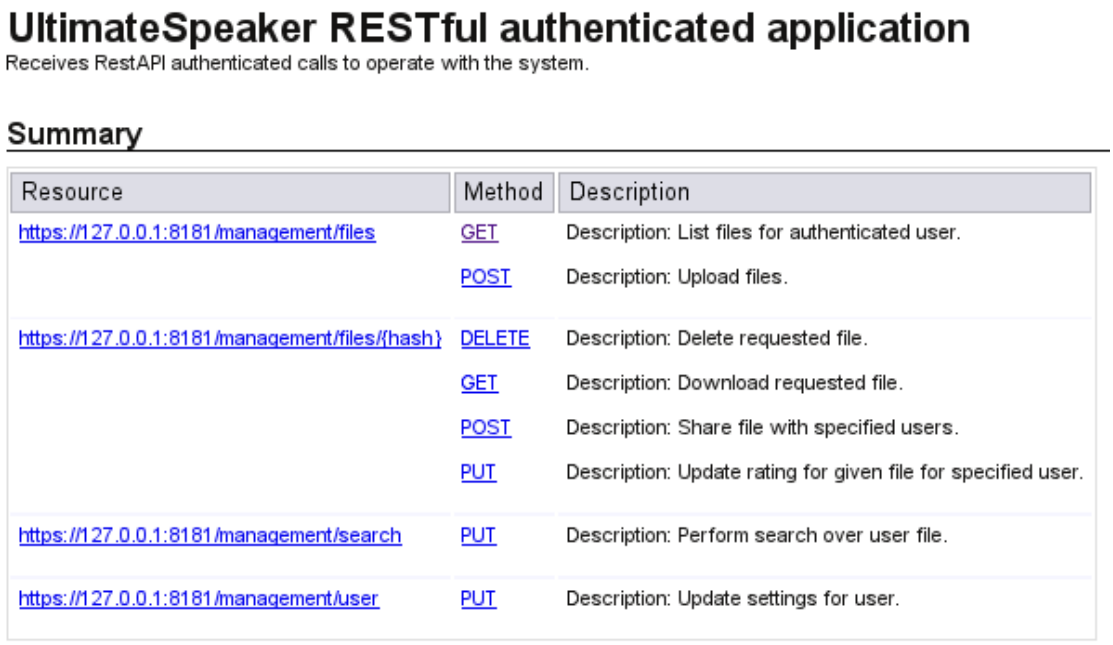
Както виждаме ресурсите са разделени на четири основни части – за операции върху всички файлове, за операции върху конкретен файл, за търсене и за промяна на потребителски настройки. За използването на всеки от тези ресурси е нужно да се включи допълнителен заглавен параметър (*Header parameter*) ***Authorization***, който се използва при базовата аутентикация на потребителите.

Ресурс за работа с всички файлове – този ресурс се използва за извличане или качване на файл. Приема заявки на следния адрес ***https://<IP>:<PORT>/management/files*** Поддържа два метода:

- *GET* метод, предоставящ списък с файловете на конкретния потребител. Поддържа параметри за извличане на резултатите по

страници. Също така можем да изберем в какъв формат да пристигне резултата, като поддържаните формати са *JSON* и *XML*, които са сравнително популярни, платформено независими езици. Избора става, чрез задаване на желания тип на съдържанието, като заглавен параметър на заявката.

- *POST* метод, за качване на файл в системата.



UltimateSpeaker RESTful authenticated application

Receives RestAPI authenticated calls to operate with the system.

Summary

Resource	Method	Description
https://127.0.0.1:8181/management/files	GET	Description: List files for authenticated user.
	POST	Description: Upload files.
https://127.0.0.1:8181/management/files/{hash}	DELETE	Description: Delete requested file.
	GET	Description: Download requested file.
	POST	Description: Share file with specified users.
	PUT	Description: Update rating for given file for specified user.
https://127.0.0.1:8181/management/search	PUT	Description: Perform search over user file.
https://127.0.0.1:8181/management/user	PUT	Description: Update settings for user.

Изображение 7: Документация на ресурсите, изискващи аутентикация

Ресурс за работа с конкретен файл – използва се за манипулиране на файлове. Достъпен е чрез заявки до адрес ***https://<IP>:<PORT>/ management/files/{hash}*.** Потребителя има право да манипулира единствено файлове, които са лично негови. Всеки файл в системата се пази чрез неговия хеш, тъй като той е уникален. Хеш-а на конкретен файл може да се вземе по време на извличане списъка на файловете за даден потребител. Поддържат се следните методи:

- *DELTE* използва се за изтриване на файла с посочения хеш. Това е необратима операция, тъй като потребителя няма кошче.
- *GET* използва се за сваляне на файла, като може да се избира дали да се свали оригиналния файл, генерирания прочит на целия файл или по слайдове(за презентационните файлове). По подразбиране се сваля прочита на целия файл.
- *POST* използва се за споделяне на файловете между потребителите. За да споделиш файл с даден потребител, трябва да знаеш неговия емайл адрес. След споделяне, файла става достъпен за упоменатите потребители.

- *PUT* използва се за даване на рейтинг за конкретния файл. Този рейтинг е споделен с останалите потребители. Тоест ако двама потребители имат един и същи файл (независимо дали той е бил споделен или просто двамата са качили файлове с еднакво съдържание), то всеки от тях ще вижда като рейтинг средно аритметичното на двете оценки. Това е направено с цел да се вижда какво мислят останалите за конкретния файл.

Ресурс за търсене – този ресурс се използва за изпълнение на различни заявки за търсене във файловете. Търсенето е единствено във файловете на потребителя, който изпълнява заявката. Заявки се изпращат на адрес ***https://IP>:<PORT>/management/search***, като единственият поддържан метод е *PUT.* Има няколко вида търсене – обикновено търсене, търсене на фрази и подсказка. Обикновеното търсене проверява за всяко съвпадение на кой да е от термините, при фразовото търсене се използват кавички за маркиране на фразата, която искаме да намерим, а при подсказките се търси само в имената на файловете - с цел бързо намиране на конкретен файл. Резултата е сортиран по ниво на съвпадение. Отново се поддържа извличане на страници.

Ресурс за промяна потребителските настройки – всеки един потребител може да настрои системата по различен начин. Тези предпочитания се пазят в базата от данни и се прилагат при различните операции. Промяната им е възможна чрез заявки на адрес ***https://IP>:<PORT>/management/user***, като единственият поддържан метод е *PUT.* Има възможност за запазване на конкретни настройки или възстановяване на стойностите по подразбиране.

Самият слой е лесно разширим и може да бъде заменен по всяко време с друга имплементация. За комуникация с потребителския интерфейс е използван протокола *HTTPS* с цел криптиране на връзката, тъй като паролите по време на регистрация се пращат в чист вид. При нужда от по-голямо натоварване, сървъра който се пуска по подразбиране може да се замени с външен такъв, като най-често използвания сървър за подмяна е *JETTY. Restlet* платформата включва разширение за този сървър, с цел лесното му интегриране. За по-голямо удобство, повечето команди могат да работят с различни типове на получаваната/изпращаната информация (*JSON*, *XML*), като желания тип се споразумява с клиента, чрез задаване на заглавни параметри на заявката. Това е направено с цел по-лесна работа с *Windows* клиенти, тъй като за тях е по-удобно да работят с *XML*, а за останалите предпочитанието е *JSON*.

5.1.2 Модул за извличане на информацията

Това е първият основен модул, част от бизнес логиката на системата. За създаването му е използван готов инструмент - ***Apache Tika***[10], предлагащ извличане на информация от много видове файлове. Самият инструмент е разширен, с цел имплементиране на онлайн презентатор, за който е нужно презентационните файлове да бъдат разделени на страници. След това за всяка страница да се изпълни процедурата по извличане на текст и генериране на прочит. Разширяването е изключително лесно, като се позволява добавяне на допълнителни типове файлове, промяна на съществуващите процесори и т.н.

За улесняване подмяната на използвания инструмент, модула е капсулиран и

се използва шаблона за програмиране „Метод Фабрика“, за скриване на конкретната имплементация на извличането на текст от файлове. Така подмяната може да стане невидимо за останалата част от кода.

От всеки файл трябва да се извлекат няколко основни неща:

- Чистия текст на файла (евентуално по страници)
- Език, на който е написан
- Допълнителна мета информацият

Извличане на чист текст – базовата функционалност на тази операция се предоставя от *Apache Tika.* С цел поддържане на по-големи по размер файлове, трябва тази операция да стане с използване на потоци и да се съхранява възможно най-малка част в паметта. Така се използва евентуални усложнения при недостиг на системни ресурси. Тъй като на презентационните файлове трябва да се обърне по-голямо внимание, за тях трябва да се разшири предлаганата функционалност, с цел извличане на текста по странично. За момента е достатъчно да се поддържат основните презентационни файлови формати – ***PPT/PTTX***. Може да се обмисли евентуално разширение с цел поддържане на презентационните файлове на отворения офис пакет (*Open Office).*

За улесняване на онлайн презентатора, всеки слайд ще представлява снимка и прикачен към нея генериран прочит. За това всяка страница от презентацията ще бъде разглеждана като отделен файл, ще се генерира изображение, после ще се извлича тескта и накрая ще се генерира прочит, на извлечения текст. Тази информация ще се запазва на файловата система и при заявка ще се изпраща на клиента.

Разпознаване на езика – нужно за последващия прочит на извлечения текст, тъй като синтезаторите за глас се нуждаят от конкретен език на гласа, който да използват. Разпознаването става на база представителна извадка от текста – един мегабайт, като се почне от началото на текста. Този размер е ориентировъчен и може да бъде променян, но върши добра работа за всички видове файлове.

Проблема с разпознатия език идва от факта, че един файл може да бъде записан на няколко езика едновременно. Тогава детектора ще разпознае езика, на който има повече написани думи. Това създава последващи проблеми при синтеза на глас, тъй като някои синтезатори не могат да прочетат дума на друг език. Такъв е примерно Английският. Тогава при грешно разпознаване, ще имаме прочит само за част от текста. Българският синтезатор, може да чете едновременно на английски и български. Това го прави идеален за такива комплексни текстове или за текст, на който езика не може да бъде идентифициран като български или английски. За това българският е зададен като език по подразбиране, при възникване на грешка по време на разпознаване.

Тук възниква въпроса, защо да не ползваме българският език и за английски текстове, тъй като синтезатора може да прочете и двата езика. Но проблема идва с думи, които са написани на неутрален език – например цифри, специални символи и т.н. Тогава английският ще бъде примесен с български, което ще

затрудни изключително много английско говорещите потребители. За това избора на език е наложителен. На потребителя се предоставя възможност за избор на статичен език, с цел използването само на английски или български синтезатор за глас. Така българските потребители могат да зададат цифрите да се четат на български, дори и в английските текстове.

Извличане на мета информация – всеки файл съдържа в себе си мета информация, която може да бъде от полза. Това е информация, различна от самото съдържание, като тип, заглавие, брой страници, автор, брой думи и т.н. Събирането на тези данни не е от съществено значение за работата на системата, но е направено с цел лесно добавяне на нови функционалности, като търсене по автор, статистическа информация за общия брой качени страници и т.н.

След като информацията бъде извлечена, тя може да бъде индексирана, върху нея може да се генерира прочит и т.н. Този модул е ключов за работата на системата и без него, никой от останалите модули не може да бъде използван пълноценно.

5.1.3 Модул за търсене

Това е вторият основен модул, час от бизнес логиката на системата. За създаването му е използвана платформата ***Elasticsearch***[29]. Основното предимство, при използването на тази система е, че тя върши голяма част от работата, а ние трябва просто да я използваме по правилния начин. За това е много важно как ще изпълним двете основни задачи при търсене – индексиране и изпълнение на заявки върху индексираните файлове. За да се избегнат грешки индексирането и търсенето се управляват от един мениджър (*ElasticsearchManager*), който се грижи за отварянето на връзка към *Elasticsearch* процеса, създаването на начален индекс, индексиране на документите и изпълнението на набор от заявки за търсене. За улесняване на разширяването на системата, този мениджър се създава използвайки шаблона за програмиране „Метод Фабрика“. Метода връща резултат от тип „*SearchManager“*, като ако искаме да използваме друга система за търсене, просто ще сменим конкретната имплементация, която фабрика метода връща.

При инициализирането на мениджъра, се отваря връзка към *Elasticserach* процеса (който се стартира автоматично, ако не е стартирал) и се проверява дали съществува подходящ индекс, за индексиране на страниците. Ако няма – такъв се създава, като за настройки се конфигурират броя на основни и второстепенни места за съхранение на информация. Тъй като *Ultimate Speaker* е настроен да работи на един компютър, системата за търсене е конфигурирана с 1 основен и 0 второстепенни, за да може да се изчислява коректно резултата, който всеки документ получава за определена заявка. При желание, системата може да бъде разширена да работи на няколко компютъра едновременно и тогава ще се промени и настройката на *Elasticsearch*-а.

Индексиране – всеки документ който бъде качен в системата бива индексиран, с цел по-бързо търсене. Индексът, който се създава , съдържа следните основни полета:

documentContent – съдържа текстовата репрезентация на файла. Поле със

стандартен анализатор, който включва “*term_vector*”, с цел последващо извличане на честотата на срещане за всеки терм (*TF-IDF*). Тази честота се използва при генерирането на резюме. Включени са филтрите за стоп думи (думи, които не се индексират) за английски и български, тъй като това са поддържаните езици от системата за момента. Включен е още филтъра *html_strip*, който изключва *HTML* таговете по време на индексиране. Това е основното поле, в което се извършват заявктие за търсене.

documentID – поле без анализ. Съдържа уникален идентификатор на индексирания документ. Съдържа майл-а на потребителя, който е собственик на файла и хеш сумата на самия файл, тъй като това са уникалните неща за всеки потребител и файл.

fileID – поле без анализ. Съдържа хеш сумата на качения файл. Използва се за предоставяне на потребителите на възможност да свалят файла от показания резултат, тъй като ресурса за сваляне приема като параметър хеш на файл.

userID – поле без анализ. Съдържа уникален идентификатор на потребителя, който е собственик на индексирания файл. Използва се за ограничаване на заявките за търсене, да са само във файловете на конкретен потребител.

documentTitle – съдържа името на файла. Използва анализатор със *n-gram* филтър, с помощта на който се изпълняват подсказки към думите за търсене. Филтърът се използва за първите 20 символа от името.

documentSummary – поле без анализ. Съдържа кратко обобщение какво има във файла. Алгоритъма за генериране на това обобщение е базиран на използване на 3-те изречения, които съдържат думи с най-висока честота на срещане (*TF-IDF*) за конкретния документ,

След индексиране се изчислява резюмето на документа, за да се избегне изчисляването му по време на търсене. Алгоритъма взема вектора на всички термини (*TermVector*) за индексирания документ и на база на него изчислява кои 3 изречения от документа са с най-висок резултат. Резултата от този алгоритъм не винаги е задоволяващ, но може да се използва за базова ориентация относно съдържанието на файла.

Търсене – това е процеса на изпълняване на заявки към индексираните документи. Имплементирани са три основни вида търсене:

- търсене на свободен текст
- търсене на фрази
- търсене за автоматично допълване на дума

Всяка от заявките трябва задължително да има съвпадение в полето за потребител, като така се ограничава всеки потребител да търси единствено в файловете, до които има достъп.

Заявките за свободен текст използват булева команда, която трябва да има съвпадение или в полето с име на файла или в съдържанието му, като при евентуално съвпадение в заглавието се повишава резултата на конкретния документ. Това се прави с цел, ако търсим за конкретна дума и тя се среща в

името на файла, то тогава е много вероятно ние да се интересуваме от този файл повече, от колкото ако съвпадението е само в съдържанието.

Фразовите заявки отново използват булева команда, но тук задължително трябва да имаме съвпадение, на маркираните фрази, в съдържанието на файла, а в последствие се изпълнява нормална заявка за свободен текст върху съдържанието и името, с цел повишаване на резултата.

Автоматичното допълване изпълнява заявки за съвпадение на фраза с префикс, но само на полето със заглавието на файла, тъй като само то се анализира от *n-gram* индекса. Избрано е само това поле, тъй като този индекс заема значително пространство на твърдия диск и би било неразумно да индексираме цялото съдържание.

Всяка от тези заявки връща също така акцентиран резултат върху полетата със съвпадение. Този резултата може да се използва за показване на допълнителна информация на потребителя, за да се прецени дали съвпадението е задоволително.

Всяка заявка приема и параметри за по странично извличане, като това много улеснява имплементацията на страници в Уеб интерфейса.

5.1.4 Модул за синтез на звук

Това е третият градивен модул, част от бизнес логиката на системата. За създаването му е използван инструмента ***Espeak***[8]. Основните предимства, на този инструмент, е че работи на всички популярни платформи, поддържа голям брой езици, предлага богат набор от настройки на генерирания глас и е изключително лесен за използване и инсталиране. Потребителя трябва да инсталира *Espeak* на компютъра, на който ще работи *Ultimate Speaker* и да включи инсталационната директория в системния път, за да може нашето приложение да достъпи външния инструмент чрез извикване на командата „***espeak***“. По време на инсталация, се задават желаните езици, които да бъдат поддържани – тук трябва да се инсталират английски „en“ и български „bg“, за да може системата да работи коректно.

За улесняване подмяната на използвания инструмент за синтез на звук, модула е капсулиран и се използва шаблона за програмиране „Метод Фабрика“, за скриване на конкретната имплементация, на генериране на прочит от текста на файловете. Така подмяната може да стане невидимо за останалата част от кода.

Позволява се сравнително голям набор от настройки на генерирания глас, част от които са описани подробно в точка 5.4 Потребителски интерфейс - (*Amplitude, Word gap, Capitals, Line length, Pitch, Speed, Encoding, Language, Markup, No final pause*). Тук ще опишем подробно останалите настройки, до които потребителя няма достъп:

- ***Read from file*** – указва на програмата името на файла, от който да бъде взет текста, върху който ще се генерира прочит. Изключително удобна опция, тъй като *Windows* операционната система има проблем с въвеждането на кирилица в командния прозорец, което затруднява

извикването на инструмента с директно посочване на текста, който да бъде трансформиран. Именно за това се използва тази настройка и текста се чете от файл, който е в подходяща кодова таблица, така че да поддържа всички символи (*UTF-8*).

- ***WAV file*** – инструктира инструмента да запише генерирания звук във файл. Използва се с цел съхраняване на прочита, тъй като без тази опция, звука се възпроизвежда след изпълнение на командата и не се записва никъде, а нашата система се нуждае от файл, който да бъде предоставен на потребителя за сваляне.
- ***Split*** – може да се използва в комбинация с „*WAV file*“ настройката, като указва след колко минути прочит да започне нов *WAV* файл. За момента тази конфигурация не се използва, но е сложена за евентуално бъдещо разширение.

Като цяло този модул предоставя достъп до възможностите на използвания инструмент за синтез на звук. Тук е имплементирана логика за изпълнение на външни команди, тъй като това е единствения модул, който не използва инструмент написан на *Java* или такъв с интерфейс за достъп.

Генерираният звук не е с перфектно качество, но генератора е с богати възможности и най-важното – безплатен е, което е задължително изискване за инструментите, които нашата система ще използва.

5.2. Модел на данните

Програмата използва както база от данни, за съхранение на малка по обем информация, така и файловата система за запис на големи и тежки файлове, които няма смисъл да бъдат съхранявани в базата от данни. Така се постига сравнително балансирано използване на ресурси, възползваме се от скоростта на изпълнение на заявките в релационните бази данни, като в същото време не се заема изключително много процесорно време със съхранението на огромни обекти в базата. Обмисля се използването на някакъв вид кеширане на записите от базата, с цел ограничаване на броя на заявките, но за сега това е само идея и не се забелязва особено забавяне при изпълнението на заявките.

Сега ще разгледаме последователно двата използвани модела за съхранение на данните – релационна база от данни и файлова система.

5.2.1. База от данни

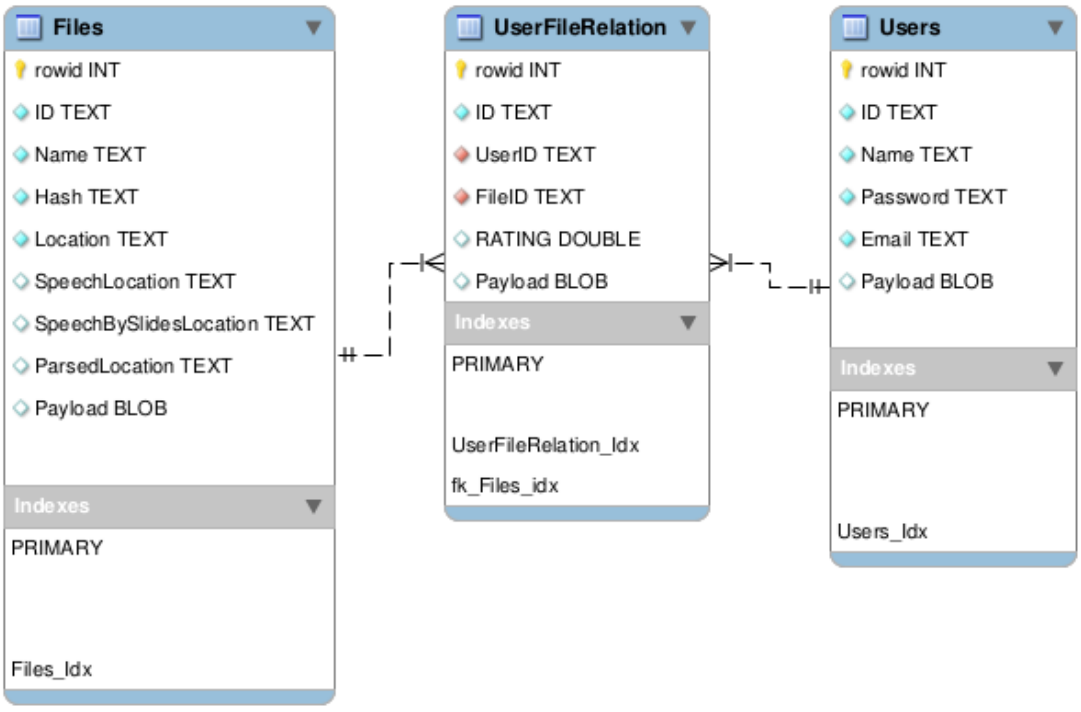
Базата от данни, която системата използва, е широко популярната ***SQLite***[25], тъй като тя изключително лесна за използване и конфигуриране, не се нуждае от предварителна инсталация и цялата база е съхранена в един файл. Тези и характеристики я правят изключително подходяща за използване в нашата система.

Отново с цел улесняване подмяната на използваната база от данни, всички нужни команди са капсулирани в един интерфейс и се използва шаблона за програмиране „Метод Фабрика“, за скриване на конкретната имплементация. Така подмяната може да стане невидимо за останалата част от кода. Мениджъра

се грижи освен за изпълнението на заявки, но също така и за връзката с базата, нейното спиране и пускане, ако се налага, създаването на потребителски пространства и всичко, което е необходимо преди началното използване и създаването на таблици и записи.

За генериране на заявките към базата не е използван външен инструмент, който да трансформира обект от кода до обект в базата, тъй като за малък продукт това би натоварило излишно разработката, а също така съществуват проблеми, с които може да се сблъскаме при използването на такива генератори, за това всички заявки са написани ръчно, на стандартен *SQL*, който трябва да работи на всички популярни бази от данни, което прави подмятана на използваната база елементарна задача.

С цел лесно разработване на кешираща стратегия, всички таблици са представени като колекции от обекти. По този начин дадена кеш стратегия може да реши колко от обектите в колекцията да бъдат пазени в паметта, кога да се прави запис в базата, кога да се извличат обекти и т.н. Така при по-интензивно използване на системата, трябва да се направи анализ на точките, в които има най-голямо забавяне и на база на този анализ да се изготви кеш стратегия, която да бъде приложена. За момента не е правен такъв тест, за това и няма приложена кешираща стратегия.



Изображение 8: Схема на базата от данни използвана от Ultimate Speaker

Схемата на базата е показана на Изображение 8. Всяка таблица съдържа три основни полета – ***rowid, ID, Payload.*** Те се използват за уеднаквяване на заявките към базата. Всяко поле се използва за следните операции:

- ***rowid*** **–** уникален идентификатор на всеки ред. Използва се за да се подсигури уникалността на отделните записи. Всяка таблица дефинира друга уникална колона или комбинация от колони, но с цел по-лесното разширяване и промяна на записите е добавено поле, което се използва единствено за гарантиране на уникалност.
- ***ID*** – всеки обект се достъпва по уникален идентификатор, но той е различен за всяка таблица (за файловете е хеш, за потребителите емайл, за връзките комбинация от хеш и майл). За това е създадено това поле, което при инициализация съдържа съответния уникален ключ. На абстрактната колекция от обекти не и е нужно да знае кой точно е ключа, за конкретната таблица, а просто използва това поле за извличане на записите.
- ***Payload*** – съдържа сериализирания обект. Използва се за конвертиране на запис обратно до *Java* обект. Това се прави с цел в таблицата да се пазят единствено полетата, които се използват в заявки или за индексиране, а не всички променливи на съхранявания обект. При запазване на конкретен обект, той се сериализира и се запазва като низ от байтове в това поле. При извличане, байтовете от полето се четат и се конструира оригиналния обект.

Сега ще разгледаме всяка таблица и ще обясним каква информация се пази в нея:

- ***Files*** **–** таблица, пазеща информацията за всеки качен файл. Тук уникален идентификатор е хеш кода на файла, който се явява и ***ID*** за таблицата. Запазва се също така името на файла, пътя до четирите вида файлове, съдържащи оригинала, извлечения текст, генерираните слайдове или генерирания прочит. Има създаден индекс върху идентификатора на таблицата с цел по-бързо намиране на желания обект. Записите се реферират в ***UserFileRelation*** таблицата, където се прави връзката кой е собственика на даден файл, като един файл може да има много собственика, за това връзката е едно към много. В таблицата не може да се съхраняват дупликатни файлове, за това ако даден потребител качи файл, който вече съществува в системата, просто се създава още една релация на този потребител със съответния файл. Така се пести време и ресурси по обработване на информация, която вече съществува в системата.
- ***Users*** **–** таблица, пазеща информация за регистрираните потребители. Уникална колона тук е емайла на потребителите. Той се явява идентификатор за таблица (*ID)*. Пазят се също така името и паролата, които са въведени при регистрация, с цел последващ вход в системата. Потребителя притежава и специфични настройки на синтезатора за глас. Те не са обособени в отделна колона, тъй като върху тях не се изпълняват заявки и те имат смисъл единствено при извличане на целия обект. За това схемата на таблицата е опростена и тези настройки се пазят единствено в полето *Payload*. Другите две полета (име и парола) са обособени в колони, за да се добави ограничение, че полетата не трябва

да са празни. Записите се реферират в таблицата ***UserFileRelation***, където се прави връзка кои са файловете, притежавани от даден потребител. Тъй като един потребител може да притежава много файлове, релацията е едно към много.

- ***UserFileRelation*** – това е таблицата, служеща за връзка между другите две таблици в системата. Тук се прави релацията между файлове и потребители. Уникалността е гарантирана на база хеш на файл и емайл на потребител. Така за всеки потребител може да се направи само една релация към конкретен файл. Тук идва и ограничението, което системата добавя, че един потребител не може да качва дупликатни файлове. В тази таблица се пази още рейтинга за даден файл от конкретен потребител. При извличане общия рейтинг на файл-а се прави заявка с агрегираща функция, която взема всички рейтинги на даден файл и им прави средно аритметично, като рейтингите със стойност 0 се пропускат. Двата външни ключа, към останалите таблици, бяха разяснени в предните два абзаца.

Вида на таблиците, генерираните заявки и евентуалните кеширащи стратегии остават невидими за останалата част от програмата. Всичко което се използва там са колекции от обекти, които поддържат всички основни методи.

5.2.2 Файлова система

Програмата се предоставя на потребителя като архив, който е поставен в определена папка. След стартиране на сървърното приложение, всички файлове се пазят в директорията, където се намира приложението. Това е направено с цел лесното премахване или преместване на програмата от един компютър на друг. Директорията за съхранение е част от настройките на администраторския интерфейс, описани в точка 5.5, и може да бъде променяна.

Създават се няколко основни поддиректории, за различните файлове, които системата съхранява:

- ***Data*** – директория съхраняваща информацията използвана от модула за търсене. Тук се пазят всички индекси, евентуално информация за други компютри, при работа с дистрибутиран индекс и т.н.
- ***Logs –*** тук се пазят логовете на системата. В основната директория има конфигурационен файл ***log4j.xml***, от където може да се настройва какви събития да бъдат записвани, какъв да е максималния размер на пазения файл, колко на брой да са пазените файлове и т.н.
- ***OrigFiles*** – както се вижда от самото име, тук се пазят оригиналните файлове, качени от потребителите. Те са непроменени и се използват при заявка за сваляне на качения в системата файл. Пътя до тези файлове, се реферира от полето *Location* в таблицата с файлове. Имената на пазените файлове се състоят от хеш сумата на качения файл. Ако той е презентационен, се запазва нов файл, името на който се състои от хеш сумата и разширение „*.slides*“, и се реферира от полето *SpeechBySlidesLocation* в таблицата с файлове.
- ***ParsedFiles*** – тук се пази извлеченото съдържание. Името на файловете

отново е хеш сумата на оригиналния файл, като така се гарантира уникалност. Пътя до конкретния файл се реферира от полето *ParsedLocation* във файловата таблица. Тези файлове се пазят, тъй като при качване на дупликатен файл от друг потребител, той не се обработва на ново, тъй като вече имаме извлечения текст и генерирания звук, но съдържанието му се индексира отново за съответния потребител. Тази операция се извършва още при споделяне на файлове с други потребители. И за да се избегне постоянното извличане на съдържанието на файловете, то се пази в тази папка.

- ***SpeechFiles*** **–** тук се съхранява генерирания прочит на съдържанието на файловете. Реферира се от полето *SpeechLocation* във файловата таблица. Използват се за отговор на потребителска заявка за сваляне на аудио запис.

Освен тези директории, на файловата система се съхраняват още няколко файла, които са нужни за правилното функциониране на програмата. Освен вече споменатия ***log4j.xml***, системата съхранява следните документи:

- ***UltimateSpeaker.cer*** **–** това е публичен сертификат, който се използва от уеб браузърите за достъпване до *RestAPI* слоя. Може да се използва за да се качи в базата от сертификати на използвания браузър.
- ***ultimateSpeaker.config*** **–** конфигурационен файл, за администраторския интерфейс на програмата. Този файл се създава автоматично, при първото стартиране на приложението.
- ***ultimateSpeaker.db*** **–** тук се намира базата от данни. Файла се създава при началното стартиране на системата, след като бъде създаден мениджъра, отговарящ за връзката с базата.
- ***UltimateSpeaker.jks*** **–** тук се съхранява частния ключ, използван при стартиране на *HTTPS* сървъра на *RestAPI* слоя.

5.2.3. Обобщение

Програмата се опитва да използва разумно системните ресурси и за това използва комбинация от база от данни и съхранение на файловата система, с цел разпределяне на натоварването. За да се ограничи използваната системна памет, не се използва никаква кешираща стратегия, но основите за евентуална такава са положени. Стремежа е да се разграничи информацията, до която се иска моментален достъп от тази, за която може да се почака известно време. Така лесно ще се идентифицира, какво да се пази в базата и какво на файловата система.

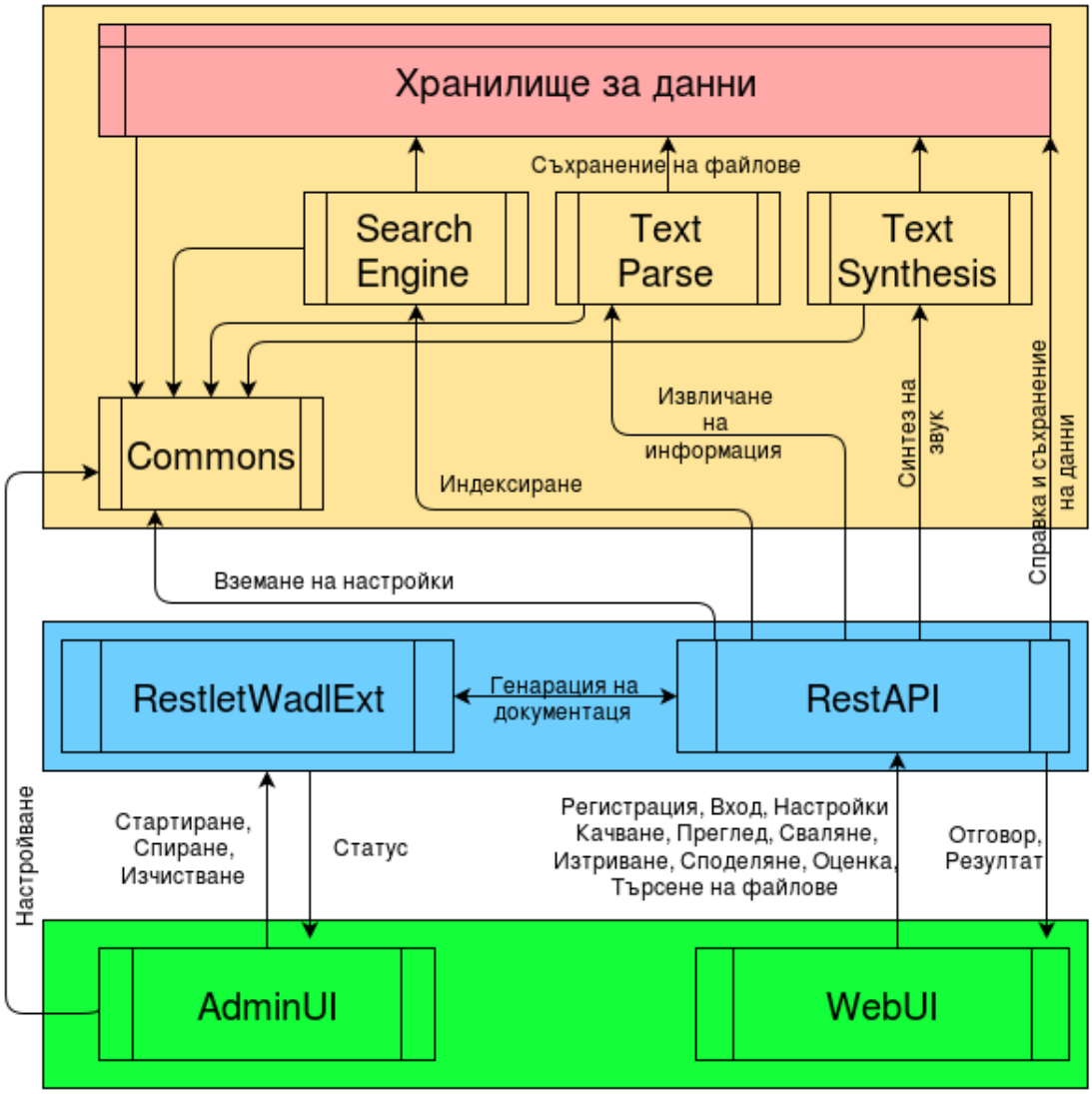
За момента тази комбинирана система на съхранение на данни върши добра работа и приложението е достатъчно производително, на база ресурсите които използва.

5.3. Диаграми

В тази точка ще покажем нагледно, какъв е потока на събитията между отделните модули, как се осъществява комуникацията между тях и цялостната

картина на приложението.

Ще започнем с графика на проектите в цялата система, на която ще се виждат нагледно комуникацията между отделните модули. След това постепенно ще детайлизираме с няколко структурни диаграми на модулите. Ще приключим с няколко графики, показващи детайлно потока на събитията при конкретни потребителски заявки.

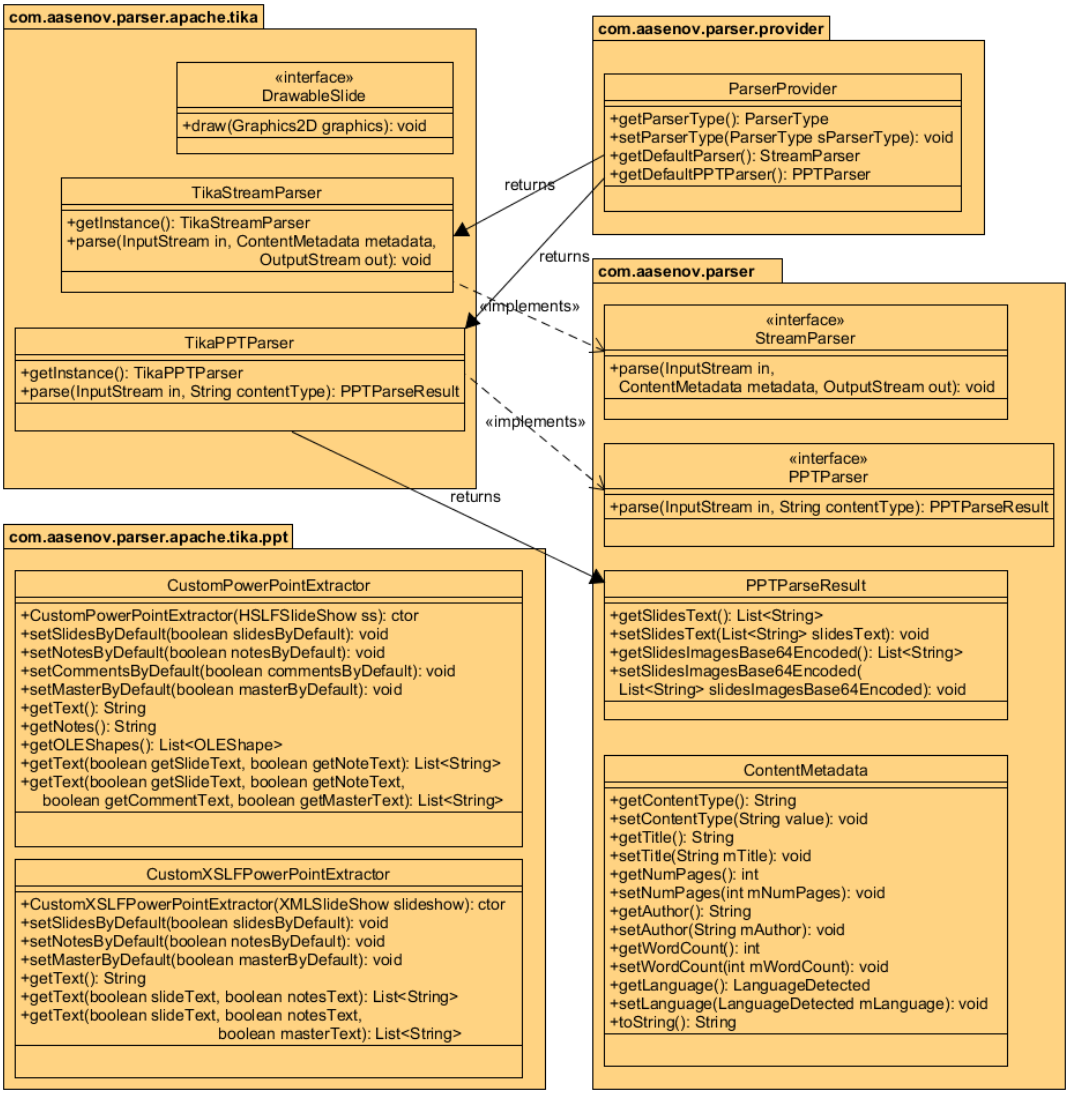


Графика 4: Организация на проектите по слоеве и комуникацията между тях

На Графика 4 се вижда нагледно организацията на проектите по слоеве и какви са основните събития, при които възниква определена комуникация между модулите. Кат започнем от интерфейсите на продукта, те комуникират единствено с контролера, който е слоя с ***RestAPI*** проектите. Единствено администраторския компонент се свързва с проект от Бизнес логиката, тъй като той прави определени настройки на системата преди самото и пускане, което значи че това не може да се случи през контролния слой.

В контролният слой се намират единствено проекти, осъществяващи комуникацията между потребителя и бизнес логиката. Там се използва и ***RestAPIWadlExt*** проекта, който е разширена версия на ***WADL*** добавката за *Restlet*, с цел по-качественото генериране на уеб документация.

В бизнес слоя имаме 4 основни проекта, всеки от които комуникира със спомагателния проект – ***Commons***, както може да се разбере и от името му. Той капсулира логиката с настройките на системата и за това всички модули имат нужда от него. Отделно от това, модулите не комуникират по между си, а само с файловата система и с контролера. Това разделение на задачите улеснява последващото разширение или евентуална подмяна на някои от модулите, тъй като те са независими един от друг. Сега ще разгледаме малко по-детайлно един от модулите в бизнес слоя на продукта.



Графика 5: UML клас диаграма на TextParse проекта, грижещ се за извличането на съдържание от текст.

На Графика 5 е показана структурата на ***TestParse*** проекта, който се грижи за извличане съдържанието на разнородни видове файлове. Както се вижда, структурата е ясно разграничена в пакети, като външните ресурси се интересуват единствено от ***com.aasenov.parser*** и ***com.aasenov.parser.provider***. **ParserProvider** класа се използва за инициализиране на правилната имплементация съответно за обикновено извличане на информация, чрез ***StreamParser***, или за извличане постранично чрез ***PPTParser.*** За улеснение последващото разширение или промяна, *ParserProvider* може да бъде настроен с тип на имплементацията, която да върне, като за момента единствения възможен тип е ***ApacheTika***.

Конструирането на правилната имплементация става в следните два „Фактори“ метода:

```
public static StreamParser getDefaultParser() {
    switch (sParserType) {
    case ApacheTika:
    default:
        return TikaStreamParser.getInstance();
    }
}
public static PPTParser getDefaultPPTParser() {
    switch (sParserType) {
    case ApacheTika:
    default:
        return TikaPPTParser.getInstance();
    }
}
```

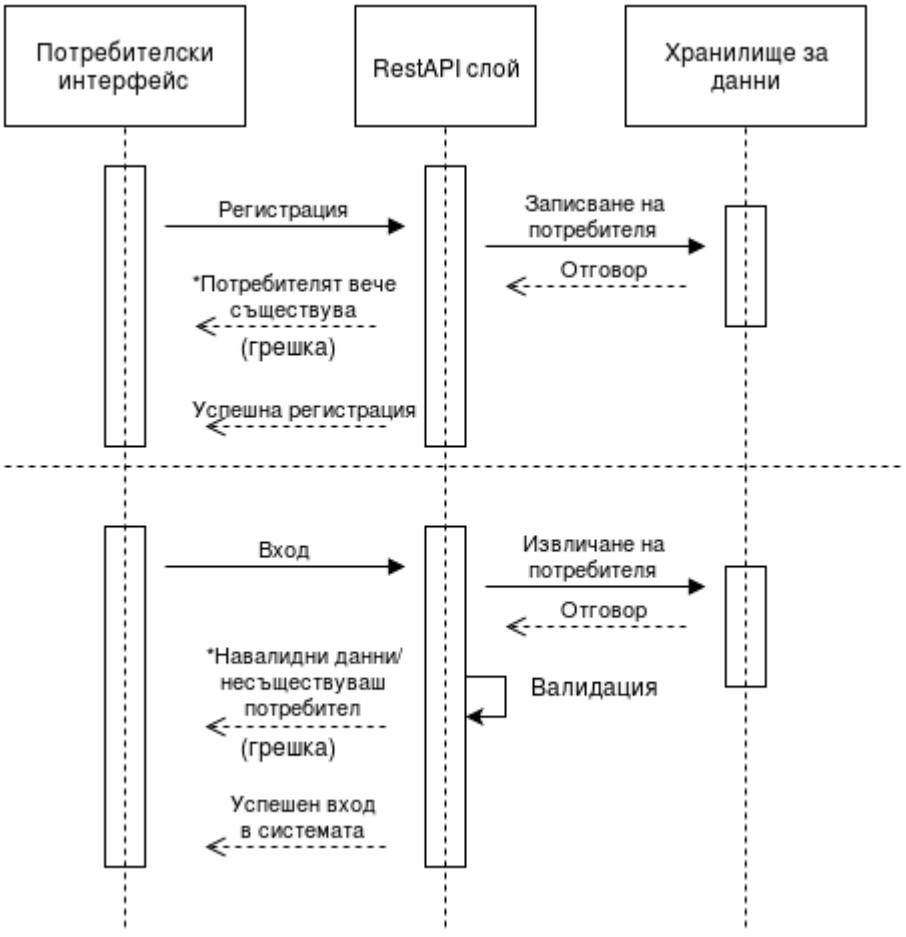
Друг интересен клас в този проект е ***ContentMetadata***. Той е независим от имплементацията, която се използва и съдържа важна мета информация, което може да бъде извлечена от всеки един файл. Тук определящо е извличането на:

- **Тип –** на файла, тъй като той се използва в последствие за определяне дали ще се продължи с постранично извличане на слайдове или не, тоест ние трябва да знаем дали това е презентация или не.
- **Език –** използва се за последващата генерация на прочит. Жизнено важен е ако потребителя не е настроил определен език, който да бъде използван за всички файлове. Без тази информация, не може да бъде пуснат правилно синтезатора.

За разделянето на презентационните файлове на слайдове, е създаден нов клас, който връща резултат от тип **PPTParserResult**, който съдържа две колекции – от съдържанието на всеки слайд и графичната му репрезентация. Тази информация се използва за последващо съхранение и връщане при заявка от потребителя.

Другите два модула – за търсене и за генерация на звук, имат подобна архитектура, тъй като модела използван при модула за извличане на звук е лесен за поддръжка и разширение (евентуална замяна на съществуващата имплементация). За това няма да се спираме подробно и няма да показваме клас диаграми на тези проекти. Вместо това ще се опитаме да визуализираме поведението на системата при изпълнението на някои стандартни операции. Първо ще започнем с някои елементарни примери, а в последствие ще покажем диаграма, включваща всички модули и слоеве на системата.

Ще започнем с описание на комуникацията между модулите и проследяване на съобщенията между тях при регистрация/вход на потребител. Може да видите на Графика 6, че при тази операция участват потребителя, контролера (който разпределя съобщенията и прави елементарна валидация) и базата от данни, където се съхранява информацията за потребителите. Комуникацията е изключително опростена и всеки модул върши единствено това, за което е направен. При евентуален проблем, потребителя бива уведомен и може да се пристъпи към последващ опит за осъществяване на заявката.

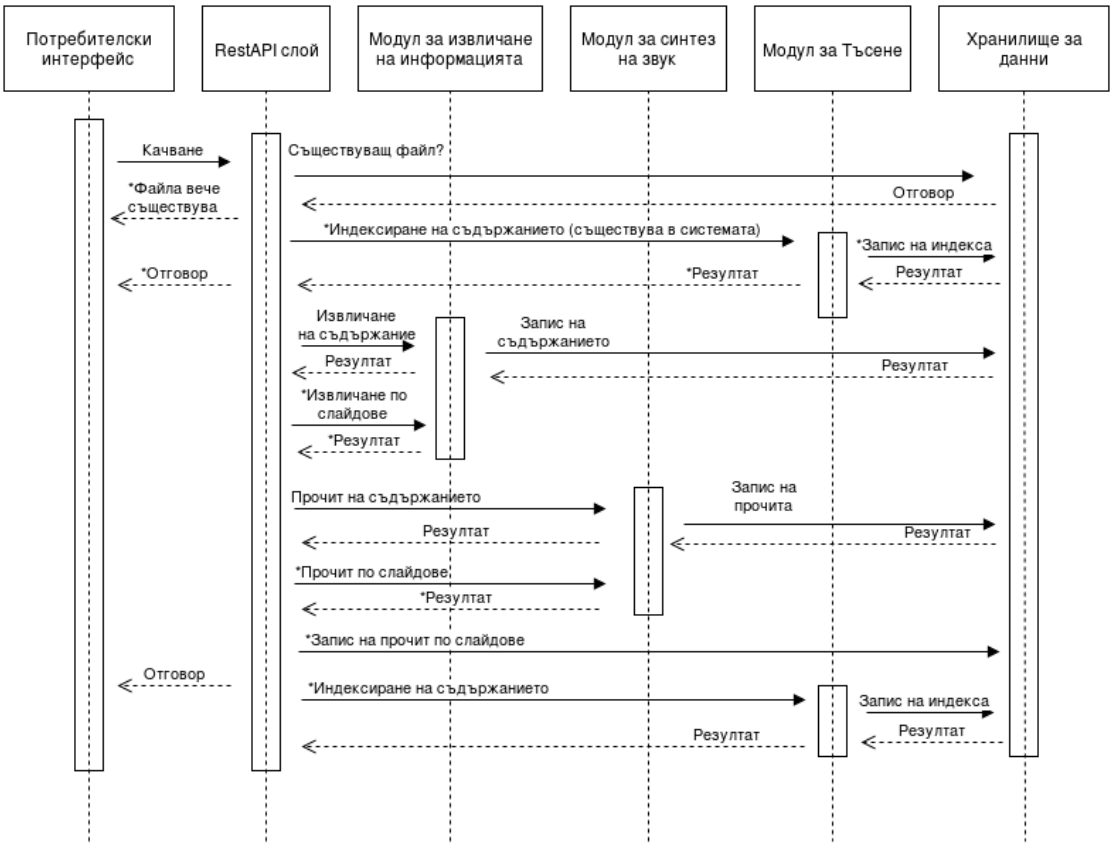


Графика 6: Поток на събитията при Регистрация/Вход на потребител

За улеснение на потребителя и да не се натоварва излишно мрежата, потребителският интерфейс прави начална валидация на данните, преди да ги

изпрати към сървъра и ако примерно сме забравили да въведем парола или емайла ни е невалиден, веднага ще се покаже съобщение за грешка, като така се ускорява обратната връзка и се избягва „спаменето“ на системата с невалидни заявки.

В следващата графика, вече ще разгледаме по-сложна операция, която включва всички основни модули на системата. На Графика 7 са показани събитията, случващи се по време на качване на файл. Както се вижда, тук вече участва цялата система, като всеки модул взема участие единствено ако се налага и за времето, което му е необходимо да извърши конкретна задача.



Графика 7: Поток на съобщенията при качване на файл

В *RestAPI* слоя е направена координацията между отделните модули и ако работата на някои от тях се провали, може веднага да се информира потребителя. Като цяло последователността на събитията е ясно очертана. Първоначално се прави проверка, дали има изобщо смисъл да се качва даден файл. Ако той съществува, за конкретния потребител, то тогава се връща грешка и процеса не продължава. Ако файла е нов за потребителя, но вече е качен от друг потребител в системата, пак се прекъсва работата и просто се създават нужните записи в базата от данни, за да се отрази че съответния потребител също има права върху дадения файл, в последствие той се индексира и с това заявката приключва.

Индексирането е асинхронен процес и не се чака приключването му, за да се върне резултата на потребителя. Тъй като това е операция, която не зависи от самия потребител и при неуспех, може да се пусне на ново, понеже разполагаме с цялата нужна информация. Освен това процеса е времеотнемащ, а предимството от скоростта, която печелим, като го пускаме асинхронно, е много по-голяма от недостатъка, че след качване на файл, в него няма да може да се търси веднага. За стартиране на индексирането се пуска следната нишка:

```
// index created file - asynchronously

new IndexThread(parsedFile.getCanonicalPath(), userFileRel.getID(), name, userID, hash).start();
```

Остана да разгледаме най-честият случай при качване - когато файлът е нов за системата. Тогава се започва пълно обработване на заявката. След като бъде записан оригинала, се изпраща заявка за извличане на информацията от ***TextParse*** модула:

```
ContentMetadata result = null;

InputStream is = null;

OutputStream out = null;

try {

    is = new FileInputStream(filePath);

    out = new FileOutputStream(resultPath);

    result = new ContentMetadata();

    ParserProvider.getDefaultParser().parse(is, result, out);

}********
```

Резултата се записва във файл, който остава в системата за последващо индексиране, а от метадатата се извлича типа и езика, за да се продължи работата. Ако файл-а е презентационен, се извиква ново извличане, но този път по слайдове. След това се продължава както при нормален файл, с извикване на модула за синтез на звук:

```
TextSynthesizerProvider.getDefaultSynthesizer(

  SynthesizerLanguage.valueOf(metadata.getLanguage().toString()),

   peechSettingsHelper.getSynthesizerSettings(TextSynthesizerProvider.getSynthesizerType(),user.getSpeechSettings()))

.synthesizeFromFileToFile(parsedFile.getCanonicalPath(), speechFile.getCanonicalPath());
```

Както се вижда, тук се използват както езика, така и настройките за конкретния потребител, тъй като той може да настрой специфичен тип генерация на глас.

След това се пуска индексиране, както беше уточнено по-горе и едновремено с това се връща отговор на потребителя.

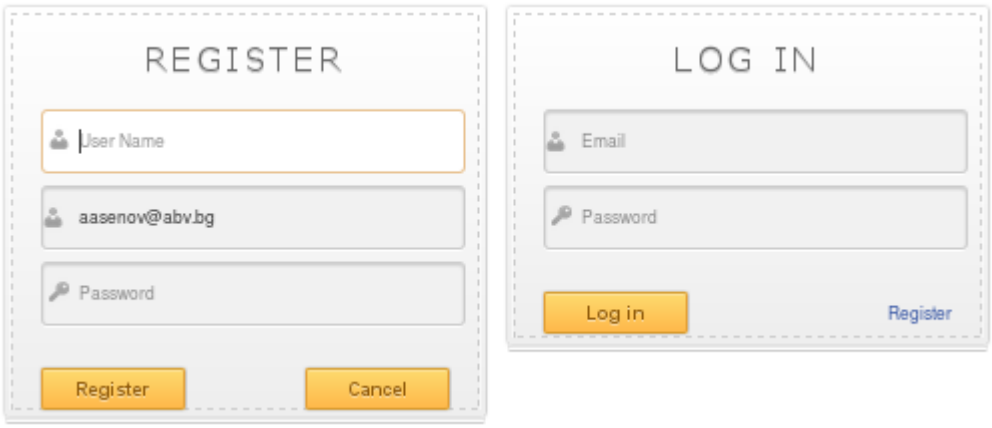
Това са в детайли, част от операциите, които изпълнява системата, но тъй като техния брой е голям, а страниците с които разполагаме – ограничени, ще се задоволим с тази малка част от общата картина. Всеки който е заинтересован,

може да прегледа проектите в кодовата база на програмата и да придобие по-задълбочена представа, как точно е имплементирана всяка една от описаните функционалности. Сега ще продължим към по-абстрактното разглеждане на системата – а именно интерфейсите които предоставя на потребителя.

5.4. Потребителски интерфейс

Потребителският интерфейс е изгледа на системата (от основната архитектура Модел-Изглед-Контролер) и в същото време клиента, от клиент-сървър модела. Той е възможно най-опростен, интуитивен и лесен за разширение. Единствената му задача е да предостави на клиента лесен достъп до функционалностите на *Ultimae Speaker*. За да няма специфични изисквания за инсталиране и благодарение на факта, че комуникацията със сървъра става чрез обикновени *HTTP* заявки, модулът ще бъде разработен като уеб страница, използвайки чист *HTML*, *CSS* и *JavaScript*. Клиентското приложение е разделено на четири основни части, описани в следващите подточки.

5.4.1. Екран за Вход/Регистрация – този прозорец трябва да позволява на потребителите да се регистрират или да влизат в системата. Това ще е началният екран, тъй като без да е влезнал в системата, клиента няма никакви права. Оформлението ще е изцяло посредством *CSS* стилове и евентуално снимки за някои фигурки. Уникалната част при регистрация е емайл адресът. Ако вече съществува потребител с такъв адрес, регистрацията ще се провали. Преди да се изпрати заявка към сървъра, се прави валидация на данните, което улеснява откриването на елементарни грешки. Прозорците изглеждат по начин, показан на Изображение 9.



Изображение 9: Екрани за Регистрация и Вход на потребител

При проблем с комуникацията към сървъра, може да се настроят различен адрес и порт, с които да се осъществи връзката. Това става от прозореца за настройки.

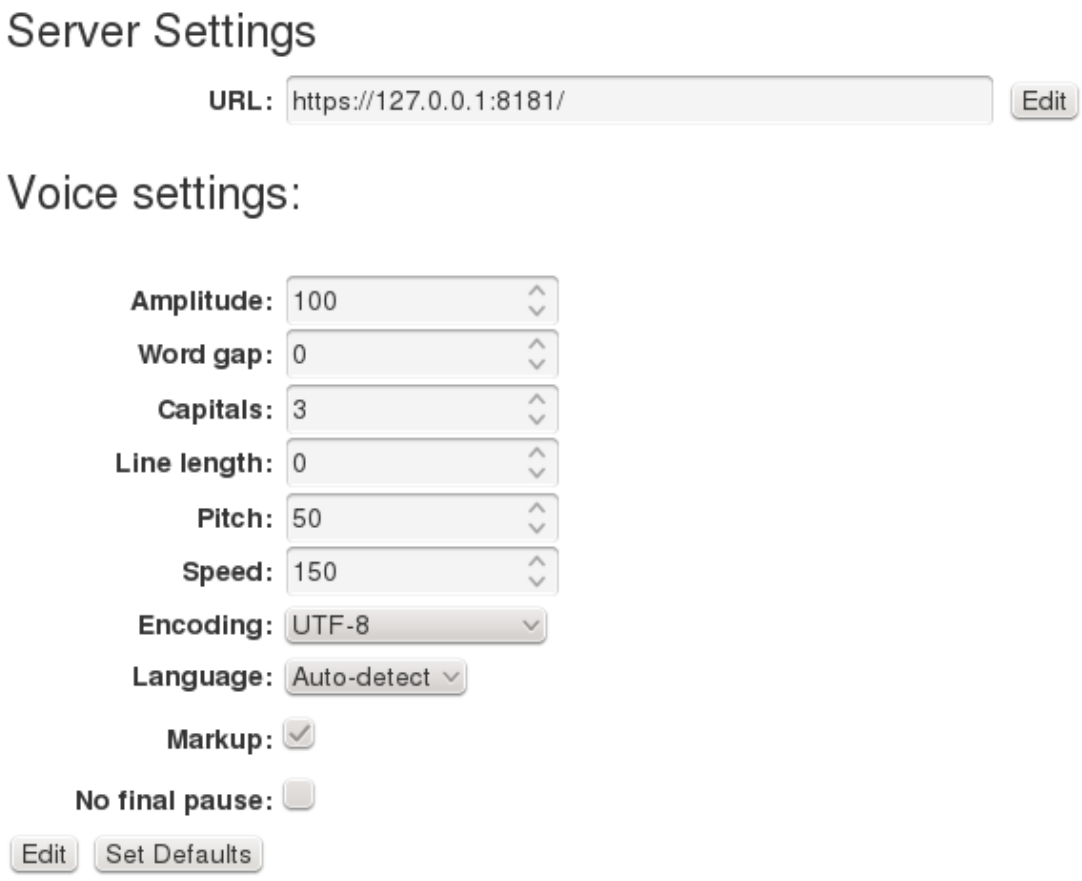
5.4.2. Екран за настройки – този прозорец съдържа конфигурациите, които могат да бъдат променяни. Съдържанието е различно, в зависимост от това дали потребителя вече е влязъл в системата или не. Ако няма потребител, с който да е

асоцииран клиентския интерфейс, видима е единствено настройката за адрес, на който да се достъпи сървъра. Това е направено, тъй като сървъра може да се намира на различно място, от последния път когато сме били в системата, а без комуникация със сървъра, клиента е безполезен.

Внимание: Адреса за достъп до сървъра е динамична настройка. Тя ще приема стойността по подразбиране (*https://127.0.0.1:8181/*) всеки път, когато страницата се презареди. За да се направи промяната постоянна, трябва да се промени стойността по подразбиране, намираща се в *ultimate-speaker.js* скрипта, ред 2:

serverSettings.serverURL = "https://127.0.0.1:8181/";

След вход в системата с конкретен потребител, екрана за настройки изглежда по начин, показан на Изображение 10.



Изображение 10: Изглед на екрана за настройки, след вход на потребител

Както се вижда, конфигурациите са много повече и позволяват фина настройка на синтезатора за глас, което е рядко срещано в продуктите от тази сфера. Всяка една конфигурация има детайлно описание, което се появява при задържане на мишката върху нейната стойност. На кратко какво представлява всяка една опция (всички се отнасят за генерацията на глас):

- ***Amplitude*** – силата на звука в интервал [0:200]. Стойността по подразбиране е 100.
- ***Word gap*** **–** положително число, изразяващо дължина на интервала между думите, по време на четене. Стойността е колко на брой 10 милисекундни интервала да се чака. При стойност 0, се взема интервала по подразбиране за всеки език. Стойността по подразбиране е 0.
- ***Capitals*** – положително число, показващо начинът, по който да се отразява срещането на главна буква в текста. Стойност **1 –** отразява се със звука „клик“. Стойност **2 –** казва се думата „главна“ преди срещането на главната буква. При стойност по-голяма от 2, се повишава тембъра на гласа, като колкото е по-голяма тази стойност, толкова по-голяма ще е разликата в тембъра. Стойността по подразбиране е 3.
- ***Line length*** **–** положително число, маркиращо дължината на линиите в текста, ако дадено линия е по-дълга, то тогава тя ще бъде разделена на две и ще бъде произнесена с пауза помежду. При стойност 0, няма ограничение от дължината на линията. Това е и стойността по подразбиране.
- ***Pitch*** **–** положително число, показващо тембъра на гласа. Колкото по-малко е числото, толкова по-нисък е тембъра и обратното. Стойността трябва да е в интервала [0:100], като по подразбиране тембъра е 50.
- ***Speed*** **–** число по-голямо от 80, показващо скоростта на четене в брой думи за минута. Минималната стойност е 80, максимална такава не съществува, но може би 500 е използваемият максимум. По подразбиране скоростта на четене е 150 думи за минута.
- ***Encoding*** – показва формата на входния текст. По подразбиране се използва *UTF-8,* като може още да се избира между 8 битов или 16 битов *Unicode* формат.
- ***Language*** **–** език за синтез, който да бъде използван. По подразбиране езика се разпознава на база на входния текст. Но ако не сме доволни от разпознаването, можем да използване тази настройка и да инструктираме синтезатора да използва конкретен език. Възможностите за избор са „Автоматично разпознаване“, „Български“ и „Английски“.
- ***Markup*** – когато е отбелязана, тази опция кара синтезатора да интерпретира *SSML(Speech Synthesis Markup Language)* и да игнорира другите тагове. По подразбиране интерпретацията е включена.
- ***No final pause*** **–** когато е отбелязана, тази опция инструктира синтезатора да не слага пауза на края на текста. По подразбиране тази пауза съществува.

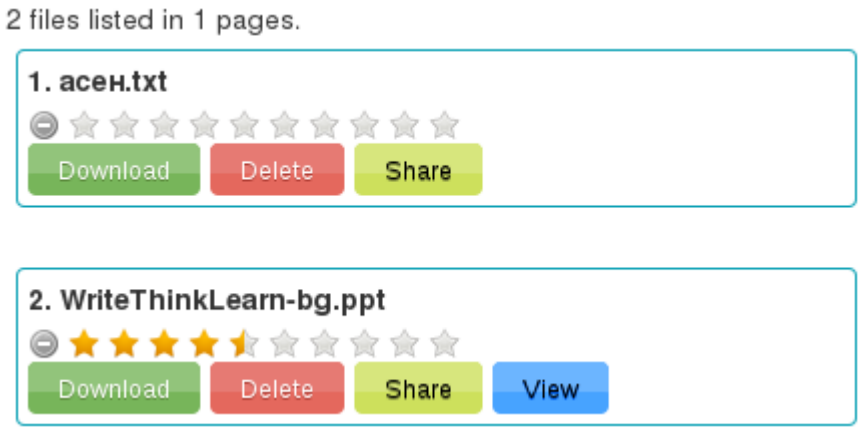
След като сме готови с настройките, идва ред на реалното използване на системата – екраните за качване, преглеждане и търсене на файлове.

5.4.3. Основен екран – след вход в системата, потребителя се пренасочва към основния екран на приложението, изглеждащ по начина, показан на

Изображение 11. Както се вижда на графиката, интерфейсът е опростен. Основната страница има две под-секции – за разглеждане на вече качените файлове и за качване на нови такива. От всяка една страница са видими навигационните бутони, което позволява на потребителя лесно да отиде на желания прозорец. Страниците, на които може да има повече информация (екраните за търсене и преглеждане на файловете) поддържат по странично извличане на информацията.

Изображение 11: Изглед на основния екран на приложението

Навигацията е „интелигентна“ и тя не извлича цялата възможна информация от сървъра. Вместо това данните са разделени на блокове от по 5 страници и за потребителя се извличат само страниците от блока, в който се намира желаната от него страница. Така се избягва натоварване на уеб браузъра и в същото време приложението е изключително бързо, тъй като не зарежда огромно количество информация.

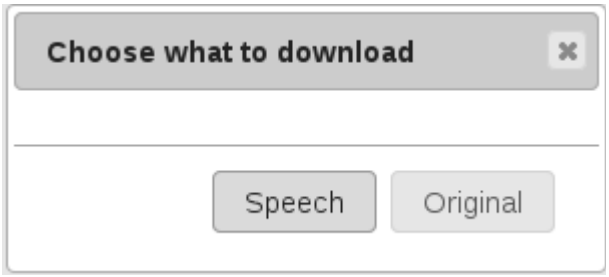


Изображение 12: Диалози показващи файловете на потребителя

Екран за преглеждане на файлове – от този екран потребителя може да преглежда всички файлове, които е качил или са били споделени с него. Начина по който изглеждат показаните файлове, е взаимстван от готовия екран при

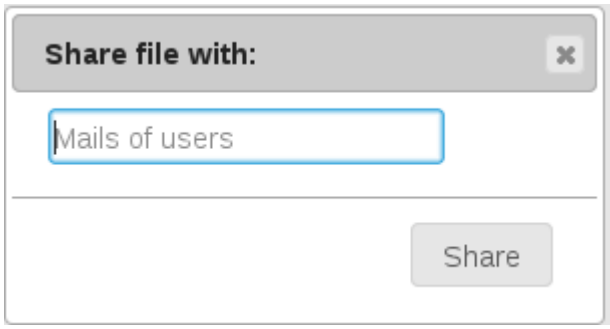
качване, използван в ***JQuery UploadFile***[19] плъгина. Визуализацията е показана на Изображение 12. Състои се от име на файл, рейтинг, и няколко бутона, които се различават в зависимост от типа на файла:

- ***Download*** – бутон за сваляне, като при натискане, се отваря диалог (Изображение 13), даващ на потребителя да избере дали да свали оригиналния файл или генерирания звук.



Изображение 13: Диалог за избор на типа файл за сваляне

- ***Delete*** – бутон за изтриване. Ако дадения файл е споделен, то операцията ще изтрие само асоциацията с текущия потребител и няма да се отрази на останалите.
- **Share –** бутон за споделяне с един или много потребители. При натискане, се отваря прозорец (Изображение 14), в който трябва да се въведе информация за емайл адресите на потребителите, с които искаме да споделим файла, разделени със запетайка. Споделянето е аналогично на качването на файл, в директорията на другия потребител, което прави по-лесно имплементирането на операцията изтриване.

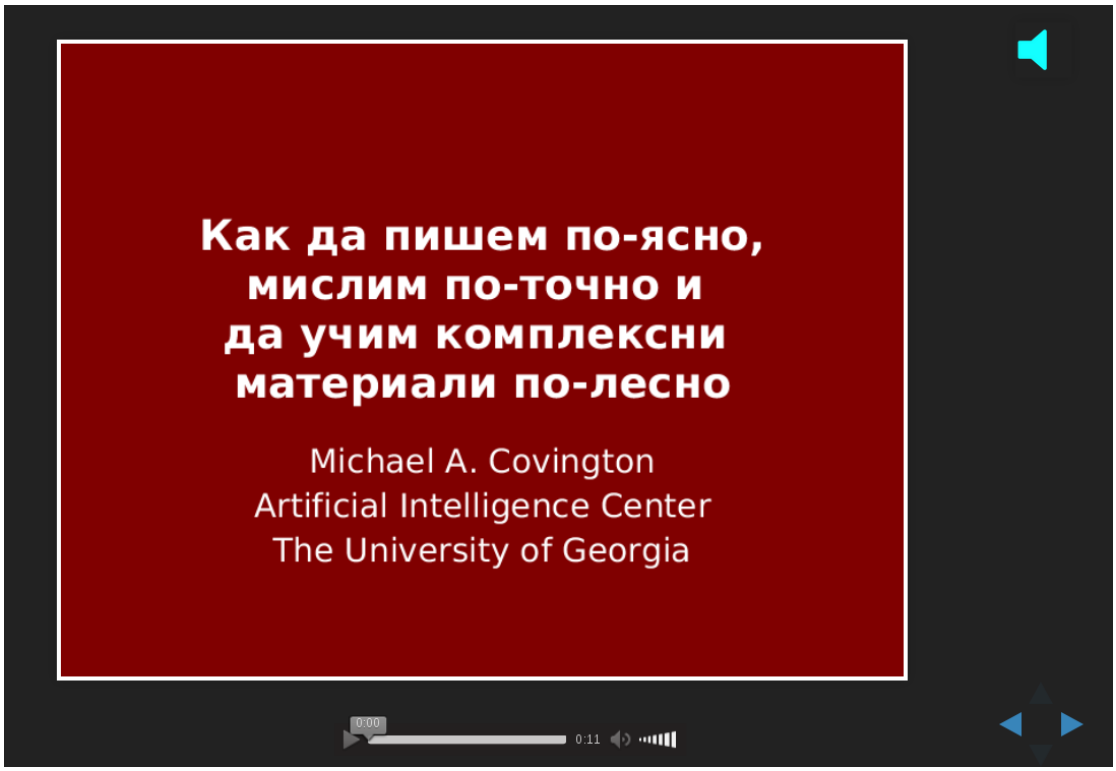


Изображение 14: Диалог за споделяне на файл с други потребители

- **Рейтинг –** Показаните звездички, под имената на файловете (Изображение 12), позволяват на потребителя да даде оценка на даден файл. По подразбиране файловете нямат рейтинг, а дори и да се даде такъв, има бутон за премахване на оценката, което е различно от оценка 0. Ако даден файл няма оценка от някой потребител, то тогава тя не се смята към общия рейтинг. Ако тази оценка е близка до 0 – то тя се смята и рейтингът на файла намалява. Оценката е споделена между всички

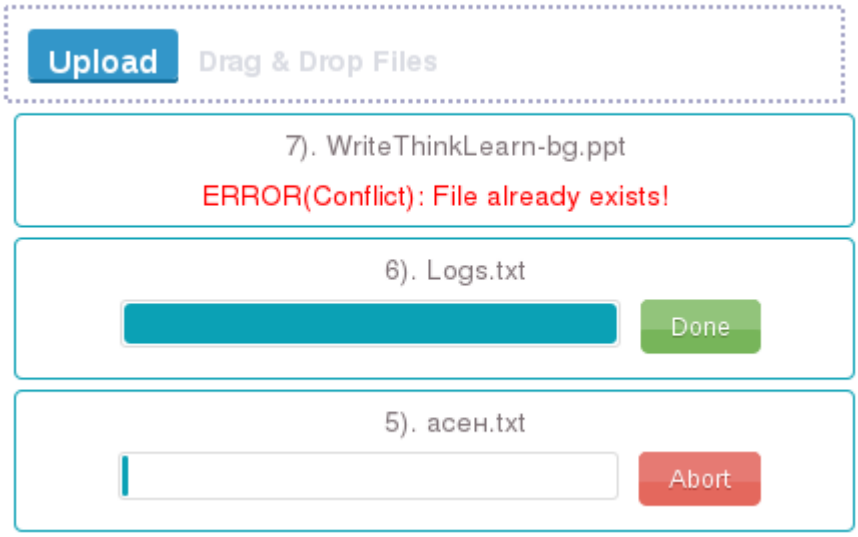
потребители, които виждат съответния файл. По време на извличане на файловете, се показва средно аритметично от оценките на всички потребители. Тази рейтингът може да се използва за преценяване качеството на лекцията или на генерирания прочит.

- ***View –*** преглед на онлайн презентация. Този бутон е видим единствено, ако съответния файл е презентационен. В този случай, има възможност за преглед на слайдовете, с вграден прочит, през уеб презентатор, изглеждащ по начина, показан на Изображение 15. Позволява се движение напред и назад по слайдовете, пускане и спиране на автоматичния прочит и други. При желание за запазване на генерираната презентация, може да се запази цялата страница в локален файл и да се прегледа в последствие. Функционалността, генерираща страницата за онлайн презентации, може да бъде видяна в Приложение 3[89]



Изображение 15: Изглед на онлай презентатора

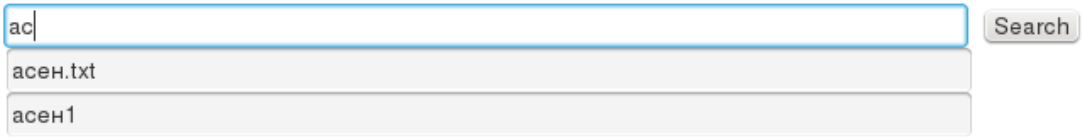
Екран за качване на файлове – той е възможно най-интуитивен и изчистен. Изглежда по начина, показан на Изображение 16. Възможни са два вида качване – чрез бутон или с провлачване. Позволява се качване на няколко файла едновременно. Има възможност да се прекрати процеса, преди той да е завършил. Уникален за всеки един файл е неговата хеш сума. Два файла са дупликатни, ако са с еднакъв хеш. Системата не позволява качване на дупликатни файлове и показва грешки при такъв опит.



Изображение 16: Изглед на екрана за качване на файлове

5.4.4. Екран за търсене във файлове – позволява търсене във файловете, които потребителя притежава. Търсенето е направено подобно на други популярни търсачки, тъй като потребителя е свикнал с конкретен изглед (примерно *http://www.google.com*). Основния екран за търсене се състои от поле за въвеждане на заявка и бутон търси. Всеки резултата представлява комбинация от следната информация за намерения файл – име и *URL* за сваляне, рейтинг, оценка за съвпадение, поле показващо съвпаденията в текста (с цел бързо преглеждане дали резултата ни върши работа) и кратко резюме на съдържанието. Показва се още колко е общия брой на резултатите от търсенето, в колко страници са разположени и колко време е отнело изпълнението на заявката. Отново не се зареждат всички резултати, а само блок от 5 страници. Поддържат се 3 вида търсене:

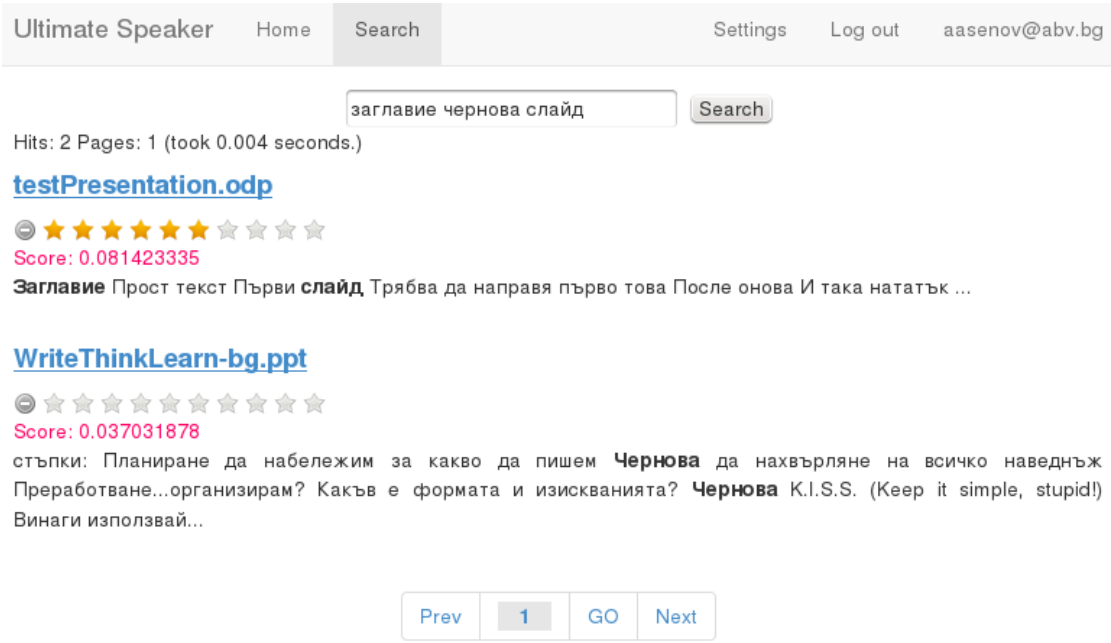
- **Автоматично допълване на думата –** това е търсене в имената на файловете. Използва се по време на въвеждане. Когато текста на заявката надхвърли 2 символа, се показват предположения за файловете, които евентуално търсим, както е показано на Изображение 17.



Изображение 17: Показване на предполагаеми думи за търсене, по време на писане

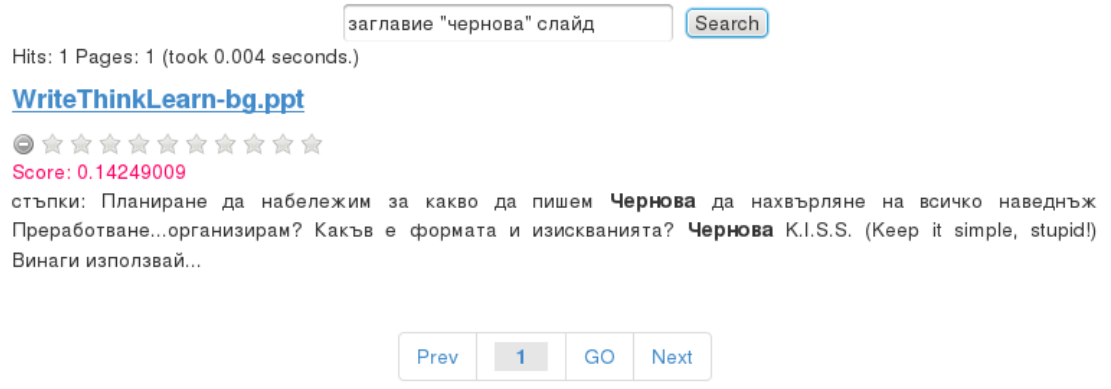
- **Обикновено търсене** – търси се съвпадение за която и да е от въведените думи. Търсенето е както в имената на файла така и в целия текст. Примерна заявка, може да се види на Изображение 18. Както се вижда, файла който съдържа по-голям брой от търсените думи е с по-голям

резултат и се показва първи.



Изображение 18: Обикновено търсене на файлове

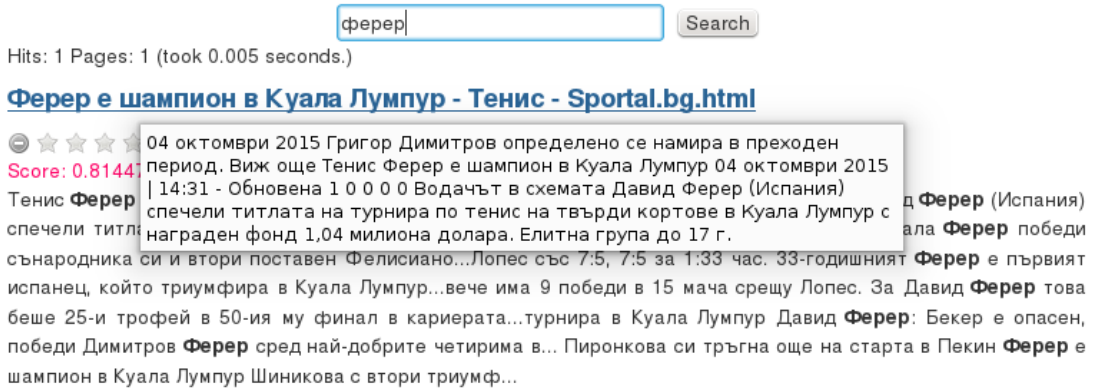
- **Търсене на фрази –** по-стриктен вид търсене. За маркиране на фраза се използват кавички. Даден файл отговаря на заявката единствено ако има съвпадение на въведените фразите. Може да се комбинират фрази с обикновени термини, както е показано на Изображение 19. Както се вижда, вече нямаме съвпадение в „testPresentation.odp“ файла, тъй като той не съдържа фразата „чернова“.



Изображение 19: Търсене на фрази, в комбинация с обикновени термини

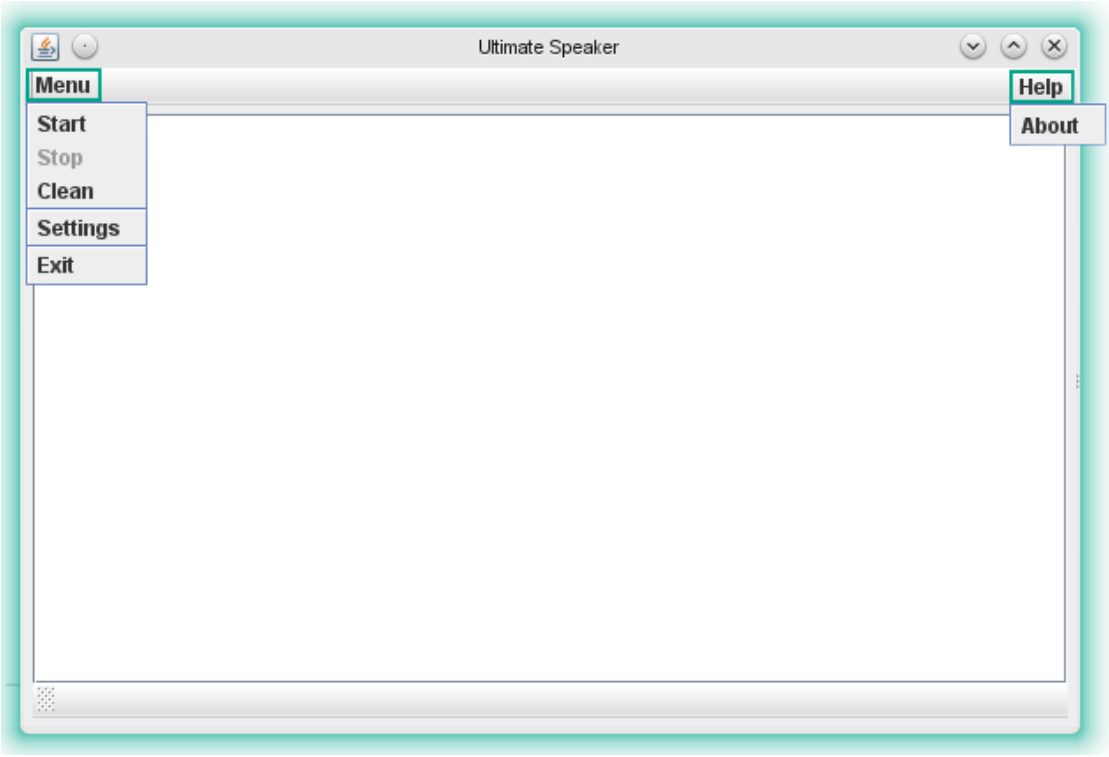
Резюмето на всеки намерен резултат може да се види, като се задържи мишката върху името на файла, както е показано на Изображение 20. Трябва да се има в предвид, че това резюме се генерира автоматично, на база честотата на

срещане на думите в даден файл. Резултата не винаги е задоволителен, но може да се използва като базова информация относно съдържанието на документа.



Изображение 20: Резюме на конкретен файл

5.5. Администраторски интерфейс

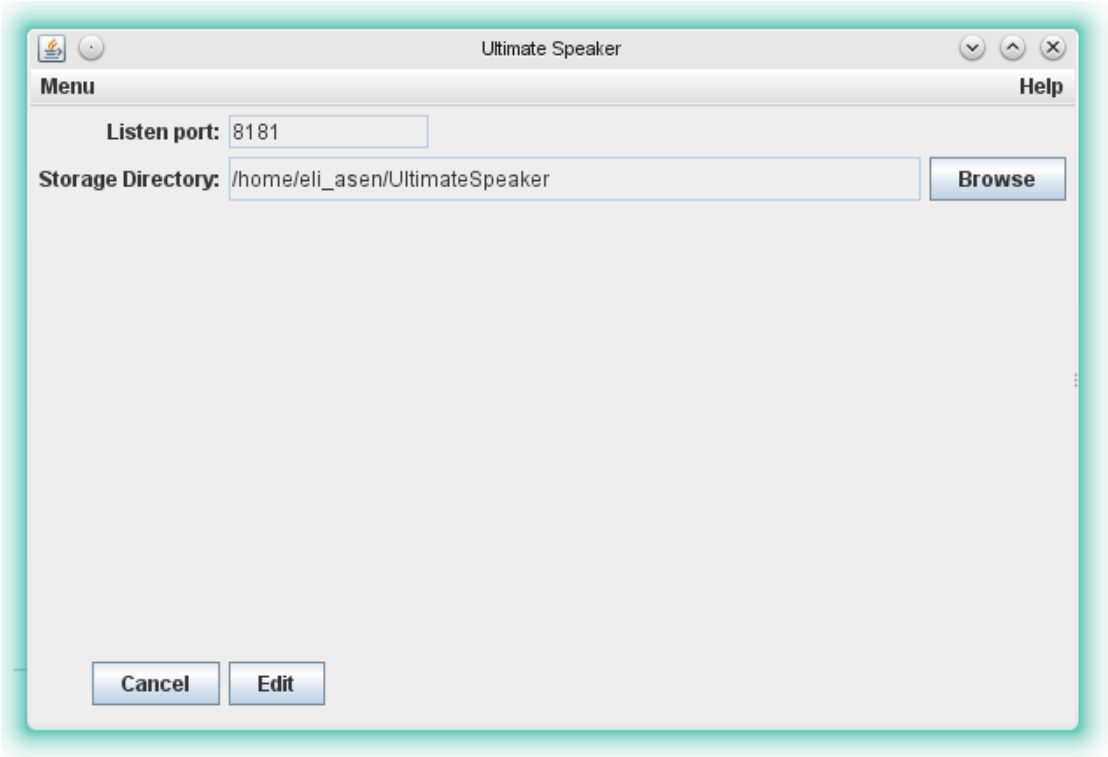


Изображение 21: Основен екран на администраторското приложение

Администратора на приложението се грижи за сървърната му част. В нея се намират контролера и бизнес логиката. Това е ядрото на програмата и без него, тя не може да функционира. Всичко което трябва да се направи за да се стартира

програмата, е да се влезе в директорията на приложението и да се стартира ***UltimateSpeaker.jar*** (под *Unix* се изпълнява командата „*java -jar UltimateSpeaker.jar*“ от командния ред). След стартиране, се визуализира екрана, показан на Изображение 21. Както се вижда, той представлява *Swing* приложение, с възможно най-изчистена и интуитивна навигация. На разположение на администратора са няколко основни бутона, с който може да се управлява цялата система.

Setting бутона се използва за конфигурация на приложението. При натискането му, се отваря екрана за настройки, които изглежда по начина, показан на Изображение 22. Позволява се промяна на порта, на който слуша контролера (*RestAPI* сървъра) и на директорията, която да се използва от системата за съхранение на файлове и данни. В зависимост от натовареността и интензитета на използване е желателно диска, на който се намира директорията за съхранение, да има достатъчно свободен капацитет.



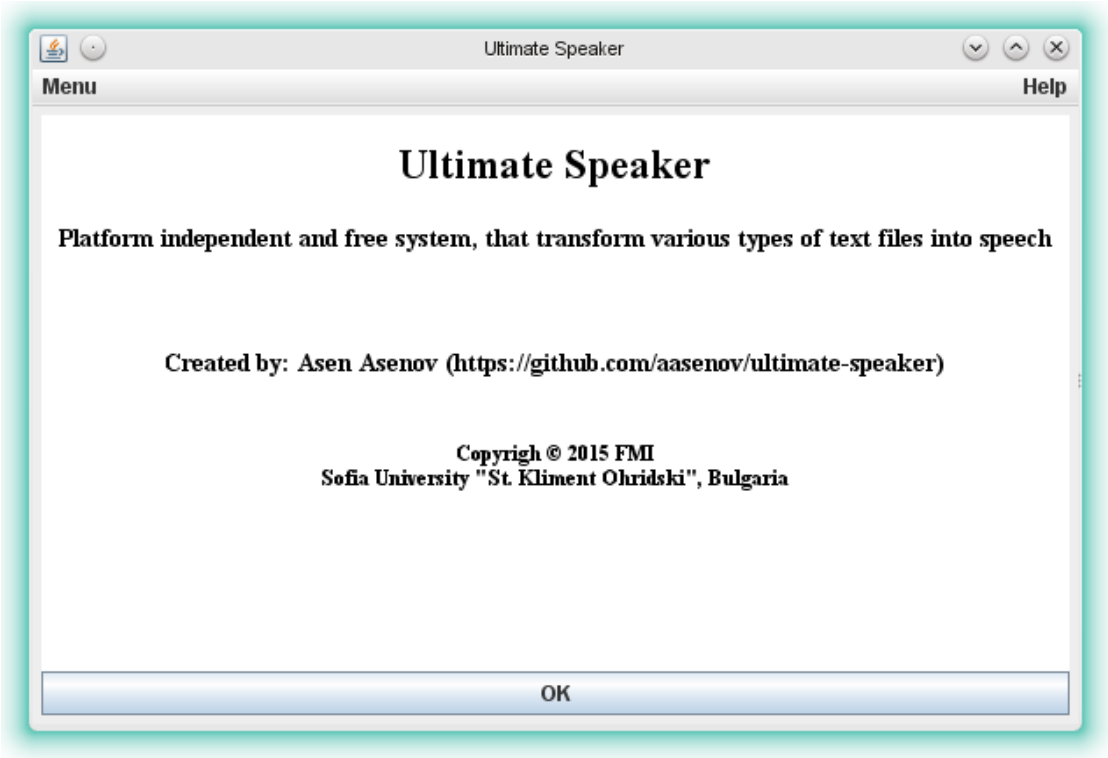
Изображение 22: Екран за настройки

Start бутона стартира приложението. При тази операция, модулите вземат зададените настройки и създават необходимите файлове за нормалната работа на системата. Свободният екран се използва за записване на събитията, случващи се по време на работа на приложението. Там може да се забележат евентуални грешки и да се придобие обща представа за работата на системата. Докато системата работи не може да се променят настройки или да се изчистват работни файлове. За да се правят каквито и да било промени, приложението трябва да е спряно.

Stop бутона прекратява работата на системата. Тази операция се изпълнява автоматично при затваряне на прозореца или при натискане на бутона ***Exit***. След като се спре работата, вече могат да се променят конфигурации или да се изчистват ненужни данни.

Clean бутона се използва за изчистване на системните файлове. При тази операция се трият всички регистрирани потребители, техните файлове и настройки, индексите за цялото приложение и базата се създава на ново. Като цяло тази операция е аналогична на нулиране прогреса на системата. Тази функционалност е изключително удобна за тестови цели и по-малко приложима в реалната среда на използване.

About бутона в ***Help*** менюто, показва екран с информация за системата - Изображение 23. Тук на кратко е описано какво точно прави системата, упоменат е създателя, онлайн проекта и мястото на създаване.



Изображение 23: Информация за приложението

Това представлява администраторския интерфейс на системата. Съдържа всичко необходимо за правилното настройване и администриране на системата. За момента работи само в графична среда, но е възможно разширяването му за приемане на команди през командния ред, тогава приложението ще може да работи и на операционни системи, без графична среда.

Глава 6. Реализация, тестване и внедряване

Тази глава ще направи задълбочен преглед на реализацията на програмата по модули, начина на тестване и тестовите сценарии, инсталационния процес и всичко останало, което не е включено в предните глави.

6.1. Реализация на модулите

Всеки един модул/слой на системата представлява отделен проект в средата за разработка *Eclipse*. Проектите са самостоятелни програмни единици и могат да се разглеждат като основни градивни единици на целия продукт. Структурата на всеки един от тях е следната:

- **Програмен код –** във всеки проект е включена папка, съдържаща програмния код. В зависимост от предназначението, папките са основно две – *src/main/java* – за всички класове, които участват активно в проекта и *src/main/test* – за всички тестови класове.
- **Използвани ресурси –** някои от проектите се нуждаят от допълнителни файлове, за правилната им работа. Това са основно конфигурационни файлове, сертификати и др. Те се намират във папка *src/main/resources*.
- **Външни библиотеки –** всяка система използва външни библиотеки, като някои от тях се общовалидни (например библиотека за запис на събитията), а други са тясно специализирани. За да се улесни структурата, има създаден отделен проект, който съхранява всички външни библиотеки. Така всеки, който се нуждае от определен външен ресурс, може да го изиска от общия проект.
- **Изграждане на програмен архив –** за компилация и пакетиране се използват ***Ant*** скриптове. Всеки проект има индивидуален скрипт, а отделно е създаден общ скрипт за цялото приложение. Той описва зависимостите между отделните компоненти и се грижи за пакетиране на финалния архив.

6.1.1 Rest API слой

Реализацията на този слой е съставена от два проекта – ***RestAPI*** и ***RestletWadlExt.***. Вторият от тях е създаден единствено за обогатяване на съществуващото разширение на *Restlet*, за генериране на документация. От оригиналния код на разширението, са променени следните файлове:

- **wadl2html.xslt –** файл, грижещ се за конвертирането на генерирания *wadl.xml* файл, към *html* страница. Това се прави при преглеждане на документацията в онлайн формат. Промяната е направена, тъй като оригиналния трансформатор не поддържаше всички възможни описания на *wadl.xml*-а.
- **OptionInfo.java –** към описанието на опциите, е добавена възможност за слагане на тип на опцията, като така при преглеждане на документацията, потребителя лесно може да разбере какъв трябва да е формата на

исканата опция.

Основният проект в *RestAPI* слоя се казва ***RestAPI***. Всички класове от този проект се намират в пакети с префикс ***com.aasenov.restapi***, като така става по-лесна ориентацията от къде идва даден клас.

В основния пакет се намират главните класове на приложението – **UltimateSpeakerAuthenticatedApplication, UltimateSpeakerBasicApplication, UltimateSpeakerComponent, UserVerifier.** Те следват стриктно архитектурата, описана при проектирането на системата. *UserVerifier* е класа, който се грижи за осигуряване сигурността на системата. В него се осъществява процеса на идентификация на потребителите, които искат да използват системата.

com.aasenov.restapi.doc **–** тук се намират всички класове, които се грижат за правилното документиране на всеки един ресурс. Това се прави с цел по-качественото документиране на ресурсите, тъй като автоматичната генерация има някои пропуски.

com.aasenov.restapi.managers – тук се намират всички обекти, които се използват за правилното разпращане на команди към другите модули. Има два основни мениджъра. ***FileManager*** - грижи се за операциите с файлове, като качване, сваляне, преглеждане, триене, споделяне, оценка. Функцията за качване на файл, може да се види в Приложение 1[78]. ***UserManager*** **–** грижи се за операциите с потребители, като създаване, промяна, извличане. Тези мениджъри са създадени с цел капсулиране на конкретни операции, с цел по-лесното им управление и промяна.

com.aasenov.restapi.mapper **–** в този пакет се намират помощните класове, за правилното определяне на типа на получената команда и очаквания резултат. Тук се прочитат заглавните полета на заявката и се определя какви типове поддържа клиента, прави се сравнение с това какви типове поддържа нашето приложение (за момента *JSON* и *XML*) и се прави избор по съвпадение.

com.aasenov.restapi.objects **–** тук са дефинирани обекти, които се използват за изпращане на колекция от резултати като отговор на заявка. Това се прави с цел, правилното трансформиране на тази колекция в текстов вид.

com.aasenov.restapi.resources **–** в този пакет са намират всички обекти, които се грижат за обработване на потребителските заявки. За всеки един поддържан *URL*, има създаден отделен обект, името на който лесно може да ни ориентира какво е неговото предназначение. Например имаме ***UsersResource***, който се грижи за операции върху всички потребители – регистрация, аутентикация. Имаме също така ***UserResourece,*** който се грижи за операции върху конкретен потребител, за което трябва да имаме вече идентифициран такъв. Такава операция е промяната на индивидуални настройки. Всеки от ресурсите следва определен дизайн. Публично достъпните методи на ресурсите са анотирани, като анотациите указват *HTTP* метода, с който трябва да се извика дадената функция. Например, във *FileResource* имаме следните публични методи:

```
@Delete("txt")
public Representation deleteFile(Representation entity)
@Get("appAll|wav|json|xml")
```

```
public Representation download()

@Put("form:txt")

public Representation rate(Representation entity)

@Post("form:txt")

public Representation shareFile(Representation entity)
```

Всяка една операция стриктно следва правилата на *HTTP* методите. *Delete* заявката се използва за изтриване на обект, като тя може да се изпълни успешно само веднъж, всеки последващ опит трябва да дава грешка за несъществуващ обект. *Get* метода връща резултат, който може да бъде кеширан от уеб браузъра, тъй като изпълнението на заявката многократно с едни и същи параметри, трябва да връща един и същ резултат. *PUT* е заявка за промяна на съществуващ обект, повторното изпълнение на която, трябва да има идентичен резултат. *POST* е метод за създаване на обект, като повторното му изпълнение трябва да дава грешка, за съществуване.

Аргументите на всеки метод, съдържат типа на заявката/резултата, който даден метод приема/изпраща. Например *@Post("form:txt")* показва, че тази команда трябва да получи заявка, в която данните са изпратени като уеб форма и връща резултат в обикновен текст. Типовете на заявката са разделени с две точки от типовете на резултата, като всеки един метод може да поддържа множество типове за конкретна посока – те се разделят по между си със знак за логическо или. Има също така методи, които дефинират само типа на връщания резултат, тъй като те не очакват определена информация от заявката – такива са *GET* и *DELETE* методите.

Това представлява накратко реализацията на *RestAPI* слоя. Няма да се впускаме в излишни подробности, тъй като всеки заинтересован може да прегледа кода на приложението – все пак той е отворен и безплатен.

6.1.2 Модул за извличане на информация

Този модул е реализиран в проекта ***TextParse***, който беше обширно описан в пакетната диаграма, от точка 5.3. За това няма да се повтаряме още веднъж в описанието на отделните пакети и копирането на код от приложението, а ще преминем към описанието на следващите модули.

6.1.3 Модул за търсене

Класовете в този модул се намират в пакета ***com.aasenov.searchengine***. Основната задача на обектите тук е да съумеят да използват възможностите на *Elasticsearch* по такъв начин, че неговата евентуална подмяна да остане незабелязана от останалите проекти. За да изпълни това условие, в основния пакет на проекта се намира интерфейса ***SearchManager***, който дефинира основните операции, които трябва да може да изпълнява всяка една конкретна имплементация, а именно:

```
public void initialize();

public void recreateIndex();

public void indexDocument(String content, String documentID, String title,
String userID, String fileID);
```

```
public void deleteIndexedDocument(String documentID);

public String generateSummary(String docID, String originalContent);

public SearchResponse searchFreeText(String query, String userID, int
from, int size);

public SearchResponse searchFreeTextAndPhrase(String freeTextQuery,
List<String> phrasesQuery, String userID, int from, int size);

public SearchResponse suggest(String query, String userID);

public void close();

public void deleteStorageFolders();
```

След това в пакета ***com.aasenov.searchengine.provider*** се намира ***SearchManagerProvider***, който използва архитектурния шаблон „Метод фактори“ за да инстанцира правилната имплементация на *SearchManager* интерфейса, която трябва се указва чрез друг обект – ***SearchManagerType.*** За момента единствената реализация е на *Elasticsearch*, но това може да се промени в бъдеще. Именно поради тази причина, единствения останал клас в проекта е в пакета ***com.aasenov.searchengine.elasticsearch***, и се казва ***ElasticsearchManager.***

6.1.4 Модул за синтез на звук

Този модул има аналогична структура на ***SearchEngine*** проекта. Всички класове са в пакет с префикс ***com.aasenov.synthesis,*** като този път интерфейса, с който се прави абстракция на конкретната имплементация, се казва ***TextSynthesizer*** и дефинира следните операции, общи за всеки един синтезатор:

```
public void synthesize(String message);

public void synthesizeFromFile(String soureceFile);

public void synthesizeToFile(String message, String destinationFile);

public void synthesizeFromFileToFile(String soureceFile, String
destinationFile);
```

Отново имаме пакет ***com.aasenov.synthesis.provider***, в който ***TextSynthesizerProvider*** обекта се грижи за връщане на правилната имплементация, която е от тип ***SynthesizerType*** и да може да синтезира на всеки от описаните езици в ***SynthesizerLanguage,*** които за момента са английски и български. За момента единствената реализация е тази, използваща *Espeak*. Именно поради тази причина, единствения останал пакет с класове е ***com.aasenov.synthesis.espeak.*** Там се намират всички класове, нужни за използването на *Espeak* инструмента. Има базова имплементация на общия интерфейс ***TextSynthesizerBase,*** и конкретни такива за отделните езици, тъй като всеки език има различни базови настройки за сила на звука, интонация и т.н. - ***BulgarianTextSynthesizer*** и ***EnglishTextSynthesizer.*** Има два обекта включващи колекция от константи (*enum).* Те се използват за по-лесното изпращане на команди към външния инструмент. ***SyntheseLanguage*** се грижи за конвертирането на езиковите константи към команди за синтезатора – например за български имаме константа *„BULGARIAN“*, трансформираща се до ***„****bg****“.*** ***SyntheseSettings*** съдържа всички възможни настройки на инструмента, като някои от тях не се използват от приложението, но могат да бъдат включени в даден момент от жизнения му цикъл. Последния обект от този пакет е

ProcessReaderTask, който се използва при изпращането на системна команда за стартиране на инструмента. Тази нишка чете резултата от изпълнението на командата.

6.1.5 Клиентски интерфейси

Потребителския и администраторския интерфейс бяха подробно обяснени в главата за проектиране. Тяхната реализация се намира в ***AdminUI*** и ***WebUI*** проектите. Няма да се спираме на подробното им разглеждане, тъй като двата проекта стриктно следват архитектурното им описание и копирането на код, няма по никакъв начин да подобри представата за тяхната реализация.

6.1.6 Модул за съхранение на данни

Този модул е реализиран в проекта ***Data*** и следва стриктно описаната архитектура. Той е един от проектите, който има написани най-много тестове, тъй като структура му позволява лесна проверка, а инструмента който се използва, е написан на *Java* и може лесно да бъде включен в тестовете. Ще разгледаме накратко структурата на модула и предназначението на някои от основните класове. Базовия пакет, в който се намират всички обекти от проекта е ***com.aasenov.database***. Там отново се намира публичен интерфейс ***DatabaseManager***, който дефинира основните функции, които всяка една конкретна имплементация трябва да поддържа, а именно:

```
   public void createTable(String tableName, String tableDeclaration, String
indexDeclaration);

   public void deleteTable(String tableName);

   public void deleteAllTableContents();

   public int getNumRows(String tableName, String whereClause);

   public <T extends DatabaseItem> void store(String tableName, Collection<T>
items);

   public <T extends DatabaseItem> List<T> select(String tableName, String
whereClause, int start, int count, String[] sortColumns, SortOrder
sortOrder);

   public double average(String tableName, String columnName, String
whereClause);

   public <T extends DatabaseItem> void delete(String tableName, String key);

   public void close();
```

В базовият пакет също така се намира друг мениджър – ***WhereClauseManager***, който се грижи за конструирането на заявки към базата, с ограничаване на резултата. Той има в себе си две колекции от параметри, едната от тях за параметрите обединени с логическо „И“ и друга за тези обединени с логическо „ИЛИ“, както се вижда от следващата извадка от конструктура на класа:

```
   public WhereClauseManager() {

           mAndCollection = new WhereClauseParameterCollection("AND");

           mOrCollection = new WhereClauseParameterCollection("OR");

   }
```

Отново имаме пакет ***com.aasenov.database.provider***, в който ***DatabaseProvider*** обекта се грижи за връщане на правилната имплементация, която е от тип ***DatabaseType*..** За момента единствената реализация е тази, използваща *SQLite,* имплементацията на която се намира в пакета ***com.aasenov.database.manager .sqlite,*** и е обект с име ***SQLiteManager.***

Друг важен пакет от проекта е ***com.aasenov.database.objects***, където се намират обектите, които се запазват в базата. Всички обекти наследяват базовият ***DatabaseItem,***в който са дефинирани общите полета за всяка таблица. Негови наследници са съответно обектите, отговарящи на таблиците от архитектурата – ***FileItem, UserItem, UserFileRelationItem.*** Друг базов обект е ***DatabaseTable,*** който представлява всяка една таблица и дефинира методи за различни операции върху редовете на таблицата. Този базов клас се използва директно за файловата и потребителската таблици. Наследява се от ***UserFileRelationDatabaseTable***, тъй като там имаме по-специфични операции за извличане на различни комбинации от редове, за конкретен потребител или файл, а не както обикновено, извличане по идентификатор.

Други важени пакети от проекта, са ***com.aasenov.database.err*** - където са дефинирани основни грешки, които могат да възникнат при работа с базата, ***org.sqlite.core*** – в който е обособен ***PreparedStatementLogWrapper*** обекта, който се грижи за проследяване и записване на всички заявки към базата.

В това се изразява накратко реализацията на всеки един от модулите, представени чрез проекти в средата за разработка. При по-голям интерес, всеки може да свали кода на приложението, от публично достъпния му адрес – ***https://github.com/aasenov/ultimate-speaker***.

6.2. Планиране на тестването

Тестването на системата е основна част от процеса на разработка. Има няколко вида тестване, които се изпълняват в определени етапи от жизнения цикъл на продукта. Ще опишем някои от основни видове тестове, които трябва да бъдат изпълнени.

***Unit* тестове –** правят се по време на писане на отделните класове.. Те трябва да обхващат възможно по-голяма част от създадените обекти. Има случаи в които тези тестове не са удачни или не отразяват реалното поведение на системата. Това са такива обекти, които пряко си комуникират с външен процес, поведението на които не може да бъде манипулирано, с цел различни тестови сценарии. В такива случаи се допуска пропускане на тези тестове, тъй като те ще бъдат обхванати в някои от следващите етапи от проверката на системата.

Модулно тестване – след разработването на всеки един модул/слой от архитектурата на системата, той трябва да бъде проверен, дали се държи както очакваме. Тъй като това са самостоятелни градивни единици на системата, не би трябвало да има проблем със самостоятелното им верифициране. Тук трябва да се отдели известно време в опит за автоматизиране на тези тестове, тъй като ръчното изпълнение на определени сценарии отнема по-малко време, но с течение на времето модулите ще трябва да се проверяват отново и отново, което ще отнема време и автоматизацията би свършила перфектна работа.

Интеграционно тестване – при комбиниране на част от модулите, трябва да се верифицира правилната им работа като група от функционалности. Изчистването на проблемите в комуникацията на ранен етап ще намали работата при последващото тестване на цялата система. Тук могат да се изпълняват част от потребителските случаи, които включват определен набор от модули.

Системно тестване – това е последната фаза на проверка на системата. Тук се проверява правилната работа на системата като цяло. Трябва да се изпълнят всички потребителски случай, да се верифицира, че всички задължителни функционални и нефункционални изисквания са спазени. Тук тестовете се изпълняват най-често ръчно, тъй като всеки сценарий се състои от множество стъпки, всяка от която трябва да се проверява щателно.

Това са четирите основни вида тестване, които ще бъдат осъществявани по време на разработването на системата. За момента може да се направи компромис единствено с автоматизацията на голяма част от тестовете. За качествено тестване се изисква почти същото време, както за разработката на самото приложение и за него може да се напише цяла отделна дипломна работа. За това модулното, интеграционното и системното тестване ще се изпълняват ръчно и единствено тестовете на програмните единици, ще бъдат интегрирани със скриптовете за компилиране и пакетиране на проекта. Така при проблем в някоя градивна единица, няма да може да се създаде финален архив и респективно ще трябва да се вземат незабавни мерки за отстраняване на грешките.

За по-лесното идентифициране и отстраняване на евентуални проблеми, по време на разработката, трябва прецизно да се записват всички събития, случващи се в системата. Всеки запис има важност (за отстраняван на грешки, информационни, предупредителни, при възникване на грешки). Информативните записи и тези с по-висока важност трябва да се показват директно в администраторското приложение. Записите за отстраняване на проблеми се записват във файл, заедно със тези с по-висока важност. Така, при евентуален срив на системата, администратора може да локализира проблема и да го отстрани по-бързо.

6.3. Модулно и системно тестване

В тази точка ще се акцентира основно върху разработването на тестовете на програмните единици, тъй като те са интегрирани в скриптовете за компилиране и пакетиране на продукта. Тези тестове могат да бъдат малко по-обширни и да тестват комбинация от обекти и работата им като едно цяла и така в частност да бъде обхванато модулното тестване.

Ще разгледаме подробно тестването на базата от данни, тъй като това е проект, който е ключов за приложението. Тестовете на останалите проекти са проектирани и разработени по аналогичен начин и за това няма да се спираме по-подробно на тях. Всички тестове на програмните единици се намират в ***src/main/test*** папката, под основната директория на всеки един проект. За ***Data*** проекта, всички тестове са с имена на пакетите, отговарящи точно на пакетите със самите обекти, а имената на тестовите класове са с имена на класа, който тестват, с думата ***Test*** накрая – например ***com.aasenov.database.provider***

.DatabaseProvider класа има тестов такъв именуван ***com.aasenov.database.provider.DatabaseProviderTest***. Тестовия клас проверява всички публични методи, на проверявания обект и евентуално чрез рефлексия функционалностите със модификатор за достъп различен от публичен. Сега ще разгледаме подробно ***DatabaseProviderTest*** класа и ще обясним важните моменти от създаването му. Ще започнем с декларацията на класа и първия му метод:

```
/** Testing {@link DatabaseProvider} functionalities. */

public class DatabaseProviderTest {

    private static Path sDataFolder;

    @BeforeClass

    public static void beforeClass() {

        // configure logger

        BasicConfigurator.configure();

         // change path helper static field in order to set jar containing
folder to temproray directory that will be

        // deleted after tests execution.

        try {

            sDataFolder = Files.createTempDirectory(null);

            // change field using reflection

                                            Field   jarContainingFolder   =
PathHelper.class.getDeclaredField("sJarContainingFolder");

            jarContainingFolder.setAccessible(true);

                                    jarContainingFolder.set(PathHelper.class,
sDataFolder.toFile().getAbsolutePath());

            // verify everything work correctly

                       assertEquals(sDataFolder.toFile().getAbsolutePath(),
PathHelper.getJarContainingFolder());

        } catch (Exception e) {

            e.printStackTrace();

            Assert.fail();

        }    }
```

В началото се декларира статичната променлива ***sDataFolder***, която се инициализира с временна директория, която да се използва за съхранение на различни програмни конфигурации, по време на тестване, и се изтрива след приключване на тестовете. Тази папка се създава преди изпълнението на тестовете, което става в метода анотиран с ***@BeforeClass.*** Тази анотация е стандартна за ***JUnit4*** и указва функция, която се изпълнява преди стартирането на тестовете. След създаването на временната папка, трябва да се накара програмата да я използва. За това се използва рефлексия, за инициализирането на статичното поле ***sJarContainingFolder*** в класа ***PathHelper***, тъй като папката, където се намира приложението, се използва като място по подразбиране, ако потребителя не укаже друго, за съхранение на данни. След като се зададе

стойност, се избягва неговата инициализация и се подсигурява, че всички компоненти ще получат времената папка, като място по-подразбиране за съхранение на данни. За да се валидира операцията по подмяна, се прави проверка: ***assertEquals(sDataFolder.toFile().getAbsolutePath(), PathHelper.getJarContainingFolder())***, при която, ако има различен от очакваното резултат, теста ще се провали. Следващият метод е с анотация ***@AfterClass***, което указва изпълнението му след приключване на тестовете и се използва за изтриване на времената папка:

```
@AfterClass
public static void afterClass() {
    // cleanup
    if (sDataFolder != null && sDataFolder.toFile().exists()) {
        try {
            FileUtils.deleteDirectory(sDataFolder.toFile());
        } catch (IOException e) {
            e.printStackTrace();
            Assert.fail();
        }
    }}
```

Неговата единствена задача е да провери, дали временната папка съществува и ако е така да я изтрие, заедно с нейното съдържание, което се прави с помощта на помощния клас от ***Apache Commons: FileUtils.deleteDirectory(sDataFolder.toFile()).*** При възникване на проблем с изтриването, теста се проваля: ***Assert.fail().*** Всички останали методи са анотирани с ***@Test***, което показва, че те реално тестват някои от методите на проверявания клас. Първо ще разгледаме тестването за промяна на типа на базата, за която се очаква да бъде създаден мениджър. Проверката се прави в следния метод:

```
@Test
public void changeDatabaseType() {
    // check default value
                assertNotNull("Default database type is null!",
DatabaseProvider.getDatabaseType());
    // check null assignment
    DatabaseProvider.setDatabaseType(null);
       assertNotNull("It is possible to assign null database type!",
DatabaseProvider.getDatabaseType());
    // check with real values
    DatabaseProvider.setDatabaseType(DatabaseType.SQLite);
         assertEquals("Unable to change database type to SQLite!",
DatabaseType.SQLite, DatabaseProvider.getDatabaseType()); }
```

Първоначално се проверява, дали има стойност по подразбиране. След това

се тества отказоустойчивостта на метода за промяна, като му се задава невалидна стойност и накрая се прави проверка с реален тип база от данни. Следва да се провери поведението на основния метод, използващ се за инициализиране на конкретен мениджър:

```
   @Test

   public void getDefaultManager() {

        // test retrieving default manager without initial setting database
type

                        assertNotNull("Default     manager     is     null!",
DatabaseProvider.getDefaultManager());

       // check retrieving exact manager

       DatabaseProvider.setDatabaseType(DatabaseType.SQLite);

                    assertTrue("Unable    to    retrieve    SQLiteManager!",
DatabaseProvider.getDefaultManager() instanceof SQLiteManager); }
```

Отново се започва с проверка, дали има стойност по подразбиране. В последствие се прави тест на реален тип база от данни. Тъй като единствената имплементация за момента е тази на ***SQLite***, проверката е единствено за нейната имплементация – ***SQLiteManager***. Накрая класа тества унищожаването на статичните променливи, които се пазят във тествания клас. Този процес е изключително важен за система, тъй като администраторския интерфейс позволява спиране и пускане на приложението, като междувременно може да се промени конфигурацията. Което значи, че спирането на програмата трябва да симулира спиране на виртуалната машина и изчистване на всички създадени обекти. Тестовия метод изглежда по следния начин:

```
   @Test

   public void destroyManagers() {

          // here we should check that instances of two invocations of
getDefaultManager should return same objects,

       // except if we didn't call destroyManagers

         assertEquals("Two invocations of getDefaultManager return different
rezults!",                              DatabaseProvider.getDefaultManager(),
DatabaseProvider.getDefaultManager());

       // check two invocations with destroy managers

       DatabaseManager manager1 = DatabaseProvider.getDefaultManager();

       DatabaseProvider.destroyManagers();

       DatabaseManager manager2 = DatabaseProvider.getDefaultManager();

       assertNotSame("Destroy managers doesn't clean static instances!",
manager1, manager2); }
```

Първоначално се проверява дали две последователни извиквания на ***DatabaseProvider.getDefaultManager()***, връщат еднакъв резултат. В последствие се слага ***DatabaseProvider.destroyManagers()***, между извличането на двата обекта и се тества дали върнатите обекти са различни. Ако това е така – то статичната променлива наистина е била заличена.

Това представлява детайлно един тестов клас. Идеята на останалите тестови обекти е аналогична и те ползват подобни методи и тактики за манипулиране и тестване на отделни компоненти от програмата. Сега ще обърнем внимание на скриптовете за компилация и пакетиране на готовия продукт, и в частност на частта с автоматичното изпълнение на тестовете на обектите.

Системата използва ***Ant*** скриптове за компилация и пакетиране. Всеки един проект има собствен скрипт, а в главната директория на продукта се намира общ скрипт, който извиква последователно скриптовете на отделните проекти и след това пакетира финалната версия на програмата – Приложение 2[85]. Сега ще разгледаме малко по-подробно скрипта на ***Data*** проекта, с тестовете на който се запознахме по-горе. Файла започва с няколко дефиниции на глобални променливи, които се използват в последствие:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project basedir="." default="Data.dist" name="Data">
    <dirname property="Data.basedir" file="${ant.file.Data}"/>
    <property name="Data.srcdir" value="${Data.basedir}/src/main/java"/>
    <property name="Data.testdir" value="${Data.basedir}/src/main/test"/>
    <property name="Data.builddir" value="${Data.basedir}/bin"/>
    <property name="jarsdir" value="${Data.basedir}/../Deploy/jars"/>
    <property name="Data.jarfile" value="${jarsdir}/Data.jar"/>
    <property               name="Commons.libsdir"               value="$
{Data.basedir}/../Commons/lib"/>
    <property             name="Commons.testlibsdir"             value="$
{Data.basedir}/../Commons/test"/>
```

Тук се дефинират местоположението на основните папки, на база текущата папка на проекта. След това се дефинира променлива, която съдържа пътя до всички класове и библиотеки, използвани при компилацията на обектите:

```
<!-- Include all dependency jars from libs and jars folders. -->
    <path id="Data.classpath">
          <pathelement location="${Data.builddir}"/>
          <fileset dir="${jarsdir}">
                <include name="**/*.jar"/>
          </fileset>
          <fileset dir="${Commons.libsdir}">
                <include name="**/*.jar"/>
          </fileset>
          <fileset dir="${Commons.testlibsdir}">
                <include name="**/*.jar"/>
          </fileset>
    </path>
```

Както виждаме, тук се включват всички вече пакетирани проекти и всички общи библиотеки, независимо че проекта не използва всички от тях. Това е направено с цел по-рядката промяна на скрипта, независимо от добавянето на допълнителни зависимости, към други библиотеки и проекти. Следва дефиницията на задачата за почистване на проекта:

```
<!-- clean target - Deletes the build directory structure -->
    <target name="Data.clean">
          <echo>Data : removing build directory</echo>
          <delete dir="${Data.builddir}"/>
    </target>
```

Тук се изтрива директорията, съдържаща стари компилирани обекти. Следва дефиниция за инициализация на работните папки и файлове:

```
<!-- init target - Create the build and dist dirs folder structures -->
<target name="Data.init" depends="Data.clean" >
  <echo>Data : creating build infrastructure</echo>
  <tstamp/>
  <mkdir dir="${Data.builddir}"/>
  <mkdir dir="${jarsdir}"/>
  <copy includeemptydirs="false" todir="${Data.builddir}">
    <fileset dir="${Data.srcdir}" excludes="**/*.launch, **/*.java"/>
  </copy>
</target>
```

Тази задача създава необходимите папки и копира нужните ресурси, в директорията за компилация. Както се вижда, тази задача има добавена зависимост от почистващата функция: ***depends="Data.clean"***. Следва функцията за компилация:

```
<!-- build target - builds the whole project -->
    <target name="Data.build" depends="Data.init" >
          <echo>Data : compiling project</echo>
          <javac encoding="utf-8" destdir="${Data.builddir}" debug="on">
                <src path="${Data.srcdir}"/>
                <src path="${Data.testdir}"/>
                <classpath refid="Data.classpath"/>
          </javac>
    </target>
```

Тук важно е да се отбележи, че се компилират както обикновените, така и тестовите класове, за да могат да се изпълнят в последствие при задачата за тестване. Следва задачата за пакетиране на ***Data.jar*** файл-а:

```
<!-- dist target - Calls build target first, then creates a jar file from
```

```
the built .class files and puts in the dist folder -->

   <target name="Data.dist" depends="Data.test" >

       <echo>Data : generating jar file</echo>

       <jar       destfile="${Data.jarfile}"       basedir="${Data.builddir}"
excludes="**/*Test.class"/>

   </target>
```

При пакетирането, се изключват всички тестови класове, тъй като те не са нужни при работата на приложението. Накрая завършваме със най-интересната част от скрипта – задачата за тестване:

```
   <!-- dist target - Calls build target first, then creates a jar file from
the built .class files and puts in the dist folder -->

   <target name="Data.test" depends="Data.build" >

       <echo>Data : executing test cases</echo>

       <junit printsummary="yes" haltonfailure="yes">

              <classpath refid="Data.classpath"/>

              <formatter type="brief" usefile="no"/>

              <batchtest>

                     <fileset dir="${Data.testdir}" includes="**/*.java"/>

              </batchtest>

       </junit>

   </target>
```

С тази операция се стартират всички тестове, които се намират в ***src/main/test*** папката на текущия проект. Изпълнението им генерира следния резултата в конзолата:

```
   Data.test:

      [echo] Data : executing test cases

      [junit] Running com.aasenov.database.provider.DatabaseProviderTest

      [junit] Testsuite: com.aasenov.database.provider.DatabaseProviderTest

      [junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
0.086 sec

      [junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
0.086 sec

   ****************************************************
```

Показва се кой тестов клас се изпълнява, колко са тестовете в него, дали има грешка при изпълнение и т.н. Хубавата част на тези тестове е, че ако има проблем при изпълнението им, пакетирането на програмата не се състой и разработчиците веднага ще забележат проблема и трябва да го отстранят.

Модулното тестване, се извършваше ръчно, при разработването на конкретен модул. Заедно с писането на тестове на обектите, се изпълняват ръчни тестове на използваните външни библиотеки, проверяват се различни начини на конфигурация и как те се отразяват на конкретния модул. Всички тези тестове са

изпълнени с цел оптимизиране на конфигурацията на конкретната функционалност и избягването на по-сериозните проблеми по време на интеграция.

Ингеграционно тестване започва след създаването на потребителския и администраторския интерфейс. Така комуникацията и стартирането на системата става по-лесно. В последствие с интегрирането на всеки един модул, се изпълняват конкретни операции, включващи неговите функционалности. Интеграцията винаги включва само 1 нов модул, с цел ограничаване на вероятните източници на проблеми. Така тестването продължава, докато не се включат всички модули, сценариите да станат пълни и системата да е готова за цялостно тестване.

За системно тестване, вече са нужни няколко човека в помощ – дипломния ръководител, няколко приятели и познати, които да изпробват програмата и основните функционалности на различни операционни системи, различни уеб браузъри, от където и изникнаха основните проблеми. Продукта притежава система за проследяване на проблемите – https://github.com/aasenov/ultimate-speaker/issues, но тъй като използването и е сравнително по-бавно от директната комуникация, системата така и не беше използвана. Отразяването на проблемите ставаше чрез директна комуникация, а оправянето им и последващото верифициране чрез преглеждане на списъка с проблеми, записан след директната комуникация. Процеса продължи, докато основните проблеми не бяха изчистени и програмата започна да работи на желаните системи и браузъри. При евентуално разширяване броя на потребителите, се предвижда започване използването на системата за проследяване на проблемите, като така може да се провери кои са текущите проблеми в системата и кога се очаква разрешаването им.

6.4. Анализ на резултатите от тестването и начин на отразяването им

Резултатите от всички тестове се отразяват в текстов файл, който е публично достъпен и се използва за начално информиране на потребителите, които искат да използват системата. Той се визуализира при отварянето на сайта на системата – https://github.com/aasenov/ultimate-speaker, и съдържа кратко описание на продукта, как протича инсталацията, поддържаните операционни системи и уеб браузъри и познатите проблеми, решението на които предстои. При добавяне на нови функционалности или оправянето на съществуващи проблеми, файла се обновява.

6.5. Експериментално внедряване

Всичко необходимо за инсталирането и пускането на системата е публично достъпно. Въпреки това ще направим кратко описание на стъпките, нужни за конфигурирането и стартирането на системата. Ще започнем с описание на изискванията за сървърната част на системата. Тя е тествана и работи с всички функционалности на следните операционни системи:

- ***OpenSuse 13.2***

- ***Fedora 19***
- ***Windows XP, 2008 R2, 7, 8***

Системните изисквания зависят от интензитета, с който ще се ползва програмата. При по-голяма натовареност, ще са нужни повече ресурси. Като базова конфигурация се изисква система със следните характеристики:

- 2GHz процесор.
- 256MB свободна RAM памет
- 250MB свободно дисково пространство (за програмата + работните файлове)

За да работи нормално сървъра, преди пускането му трябва да са изпълнени следните стъпки:

- Инсталиране на последната *Java 1.7* виртуална машина – 1.7.79/1.7.80, за съответната операционна система.
- Инсталиране на инструмента за синтез на звук ***ESpeak***[10], за съответната операционна система, със езиците български и английски (**„bg“** и **“en“**). Директорията къде се намира изпълнимият файл на синтезатора, трябва да е включен в системния път, за да може сървърът да го стартира.

След като се изпълнят тези стъпки, единственото което трябва да се направи, за стартиране на приложението, е да се изпълни ***UltimateSpeaker.jar*** файла от главната директория на продукта.

Клиентското приложение е проверено на следните уеб браузъри:

- ***Firefox 41.0+***
- ***Chrome 44.0+***

То не се нуждае от допълнително ресурси, така че за него няма повече системни изисквания. За да може да се осъществи връзката със сървърната част, която използва криптиран канал на комуникация, уеб браузърът трябва да приеме сертификата на сървъра, който е ръчно подписан и не се приема като сигурен. Това е така, тъй като сертификата е безплатен и подписването му с валидно ***Certificate Authority***, изисква средства. За да приеме браузъра сертификата, трябва да се инсталира ***UltimateSpeaker.cer*** файла, в уеб браузъра и да се приеме за адреса на сървъра. По-лесният вариант за изпълнението на тази операция е, преди да се отвори клиентското приложение, да се достъпи *RestAPI* документацията на сървъра, на адрес *https://<ip>:<port>/api-docs*. Така уеб браузъра ще пита изрично дали да приеме сертификата. След като се изпълнят тези стъпки, може да се стартира клиентското приложение от файла ***UI/index.html*** и да се започне използването му.

Глава 7. Заключение

7.1. Обобщение на изпълнението на началните цели

Разработката на системата протече почти два пъти по-дълго от очакваното, но резултата от изпълнението и е повече от задоволителен. Всички начално поставени цели са изпълнени и проверени на поне две операционни системи. Сървърната част е качена на публичен сървър с цел дългосрочно тестване преди пускане на финалната версия. Крайният облик на *Ultimate Speaker* отговаря на поставените изисквания и не отстъпва по нищо на другите подобни системи. Има дори аспекти в които нашата система се държи доста по-добре от останалите и предоставя удобни функционалности, които все още не са достъпни в никоя съществуваща система, от този тип.

7.2. Насоки за бъдещо развитие и усъвършенстване

Продукта поставя здрави основи за развитие на една модерна и лесна за използване система за прочит на файлове. Тя може да се развива в много посоки и само желанията на потребителите могат да подредят по важност списъка с възможните подобрения. Основните насоки за развитие, могат да се групират в следните точки:

- Увеличаване броя на поддържаните езици и като цяло усъвършенстване на синтезатора за глас.
- Усъвършенстване на потребителския интерфейс, евентуално създаване на теми, превод на различни езици, създаване на алтернативни клиенти и др.
- Конфигуриране и инсталиране на системата в дистрибутиран вид, което ще увеличи броя на поддържаните конкурентни заявки. Като цяло разширяване на модела на инсталация.
- Усъвършенстване и добавяне на нови файлови формати, разширяване броя на поддържаните презентационни файлове, генериране на различни видове звукови файлове – подобряване работата с различните файлови формати.
- Структуриране и изграждане на система за автоматично тестване на потребителските случай. Това е огромно разширение, което може да подобри неимоверно процента на открити проблеми и да ускори процеса по разширяване на системата. В даден момент дори ще стане невъзможно разширението без тези автоматизирани тестове, тъй като никога няма да сме сигурни какво работи и какво не.

Системата използва изключително много технологии и външни библиотеки, които могат да бъдат променяни и усъвършенствани. Този процес е продължителен и времетраене и единствено увеличаване броя на разработчиците и тестващите, може да го направи по-плавен и стабилен. За това разширяването на продукта идва с разширяване на екипа, който го поддържа.

Използвана литература

[1]"Google Translate." Google Translate. N.p., n.d. Web. <https://translate.google.com/>

[2]"YakiToMe!" YakiToMe! N.p., n.d. Web. <https://www.yakitome.com>.

[3]"iSpeech." iSpeech. N.p., n.d. Web. <http://www.ispeech.org/>.

[4] "SpokenText." Online Text to Speech (TTS) Converter. N.p., n.d. Web. 11 Sept. 2015. <http://www.spokentext.net/>.

[5]"Тестове: Синтезатор SpeechLab („Гергана")." Тестове: Синтезатор SpeechLab („Гергана"). N.p., 23 Apr. 2014. Web. 25 Feb. 2015. <http://testove.nllb.bg/mod/page/view.php?id=25>

[6]"Тестове: Синтезатор Innoetics Irina („Ирина“)." Тестове: Синтезатор Innoetics Irina („Ирина“). N.p., 23 Apr. 2014. Web. 25 Feb. 2015. <http://testove.nllb.bg/mod/page/view.php?id=26>

[7]"Тестове: Синтезатор eSpeak (и с български глас)." Тестове: Синтезатор eSpeak (и с български глас). N.p., 23 Apr. 2014. Web. 25 Feb. 2015. <http://testove.nllb.bg/mod/page/view.php?id=27>

[8]"ESpeak Text to Speech." ESpeak: Speech Synthesizer. N.p., n.d. Web. 01 Mar. 2015. <http://espeak.sourceforge.net/>

[9]"Apache POI - the Java API for Microsoft Documents." Apache POI - the Java API for Microsoft Documents. N.p., n.d. Web. 12 Mar. 2015. <http://poi.apache.org/>

[10]"Apache Tika - Apache Tika." Apache Tika - Apache Tika. N.p., n.d. Web. 09 Mar. 2015. <http://tika.apache.org/>

[11]"Jersey." Jersey. N.p., n.d. Web. 25 Mar. 2015. <https://jersey.java.net/>

[12]"RESTEasy - JBoss Community." RESTEasy - JBoss Community. N.p., n.d. Web. 25 Mar. 2015. <http://resteasy.jboss.org/>

[13]"Restlet API Platform." Restlet. N.p., n.d. Web. 25 Mar. 2015. <http://restlet.com/>

[14]"Bootstrap · The World's Most Popular Mobile-first and Responsive Front-end Framework." Bootstrap · The World's Most Popular Mobile-first and Responsive Front-end Framework. N.p., n.d. Web. 23 Sept. 2015. <http://getbootstrap.com/>

[15]"JQuery." JQuery. N.p., n.d. Web. 23 Sept. 2015. <https://jquery.com/>

[16]"JQuery UI." JQuery UI. N.p., n.d. Web. 23 Sept. 2015. <https://jqueryui.com/>

[17]"JQuery Form Plugin." JQuery Form Plugin. N.p., n.d. Web. 23 Sept. 2015. <http://malsup.com/jquery/form/>

[18]"JQuery Raty." JQuery Raty. N.p., n.d. Web. 23 Sept. 2015. <http://wbotelhos.com/raty>

[19]"Callbacks." JQuery Upload File Plugin Demo. N.p., n.d. Web. 23 Sept. 2015. <http://hayageek.com/docs/jquery-upload-file.php>

[20]"Hakimel/reveal.js." GitHub. N.p., n.d. Web. 23 Sept. 2015. <https://github.com/hakimel/reveal.js/>

[21]"Impress/impress.js." GitHub. N.p., n.d. Web. 23 Sept. 2015. <https://github.com/impress/impress.js>

[22]"Fmi/www-lectures." GitHub. N.p., n.d. Web. 23 Sept. 2015. <https://github.com/fmi/www-lectures>

[23]"Deck.js." » Modern HTML Presentations. N.p., n.d. Web. 23 Sept. 2015. <http://imakewebthings.com/deck.js/>

[24]"Robflaherty/html-slideshow." GitHub. N.p., n.d. Web. 23 Sept. 2015. <https://github.com/robflaherty/html-slideshow>

[25]"SQLite Home Page." SQLite Home Page. N.p., n.d. Web. 07 Apr. 2015. <https://www.sqlite.org/index.html>

[26]"Bitbucket." Xerial / Sqlite-jdbc —. N.p., n.d. Web. 07 Apr. 2015. <https://bitbucket.org/xerial/sqlite-jdbc/>

[27]"Welcome to Apache Lucene." Apache Lucene -. N.p., n.d. Web. 25 Sept. 2015. <https://lucene.apache.org/>

[28]"Solr Is the Popular, Blazing-fast, Open Source Enterprise Search Platform Built on Apache Lucene™." Apache Solr -. N.p., n.d. Web. 25 Sept. 2015. <http://lucene.apache.org/solr/>

[29]"Elasticsearch: RESTful, Distributed Search & Analytics | Elastic." Elasticsearch: RESTful, Distributed Search & Analytics | Elastic. N.p., n.d. Web. 25 Sept. 2015. <https://www.elastic.co/products/elasticsearch>

[30]"Web Application Description Language." Web Application Description Language. N.p., n.d. Web. 14 Oct. 2015. <http://www.w3.org/Submission/wadl/>

Приложения

Приложение 1: Функция за обработване качването на файлове

```
 /**
    * Method that handle file upload.
    *
    * @param fileItem - item to be uploaded.
    * @param userID - ID of user that this file belongs to.
    *
    * @return ID of file that was uploaded, <b>Null</b> if such doesn't exists.
    * @throws Exception in case of error.
    */
   public String handleFileUpload(org.apache.commons.fileupload.FileItem fileItem, String userID) throws Exception {
       String result = null;
       String name = fileItem.getName();
       if (name == null) {
           sLog.error(String.format("Unable to determine file name of %s='%s'. Skipping.", fileItem.getFieldName(), new
String(fileItem.get(), StandardCharsets.UTF_8)));
       } else {
           // store in FS
           File file = new File(mOriginalFilesDir, name);
           if (file.exists()) {
               // generate random string for duplicate files. If their hashes match this file will be deleted.
               file = new File(mOriginalFilesDir, name + UUID.randomUUID());
           }
           if (sLog.isDebugEnabled()) {
               sLog.debug(String.format("Storing original file in '%s'.", file.getCanonicalPath()));
           }


           // compute md5 checksum during file upload to prevent reading file twice.
           String hash = "tempHash";
           InputStream fis = null;
           OutputStream out = null;
           byte[] buffer = new byte[STREAM_READ_SIZE];
           try {
               fis = fileItem.getInputStream();
```

```
    out = new FileOutputStream(file);

    MessageDigest md = MessageDigest.getInstance("MD5");
    int numRead;
    do {
        numRead = fis.read(buffer);
        if (numRead > 0) {
            // write file to FS
            out.write(buffer, 0, numRead);
            // compute hash
            md.update(buffer, 0, numRead);
        }
    } while (numRead != -1);

    hash = new BigInteger(1, md.digest()).toString(16);
    result = hash;
} finally {
    if (fis != null) {
        fis.close();
    }
    if (out != null) {
        out.close();
    }
}
if (sLog.isDebugEnabled()) {
    sLog.debug(String.format("Storing original file in '%s' was successful.", file.getCanonicalPath()));
}

// store in database
UserFileRelationItem userFileRel = new UserFileRelationItem(userID, hash);
FileItem existingFile = mFilesTable.get(hash);
if (existingFile == null) {
    if (sLog.isDebugEnabled()) {
        sLog.debug(String.format("Storing file with has '%s' in database.", hash));
    }
    // ranam in order to store files by hash
    File newFile = new File(mOriginalFilesDir, hash);
```

```
            if (file.renameTo(newFile)) {
                file = newFile;
                FileItem newDBFile = new FileItem(name, hash, file.getCanonicalPath(), null, null, null);
                try {
                    mFilesTable.add(newDBFile);
                    mUserFileRelTable.add(userFileRel);
                } catch (Exception ex) {
                    // Probably another thread already crate file with same hash
                    sLog.error(ex.getMessage(), ex);
                    existingFile = mFilesTable.get(hash);
                }
            } else {
                sLog.error(String.format("Unable to rename file %s to %s. Skip uploading.",  file.getCanonicalFile(),
newFile.getCanonicalPath()));
                file.delete();
                return null;
            }
        }

        if (existingFile != null) {
            // check whether this file exist for current user
            UserFileRelationItem existingReletaion = mUserFileRelTable.get(userFileRel.getID());
            if (existingReletaion == null) {
                if (sLog.isDebugEnabled()) {
                    sLog.debug(String.format(
                            "File with has '%s' already exists in database. Just add access for user '%s'", hash,  userID));
                }
                // this user has no access to this file, add one
                try {
                    mUserFileRelTable.add(userFileRel);
                } catch (Exception ex) {
                    sLog.error(ex.getMessage(), ex);
                    result = null;
                }
            } else {
                // do not allow duplicate files
                sLog.error(String.format( "File with hash '%s' already exists. Existing file name is '%s'. Skipping file
upload!",  hash, existingFile.getName()));
```

```
                result = null;
            }
        }

        if (existingFile != null) {
            if (sLog.isDebugEnabled()) {
                sLog.debug(String
                        .format("File with has '%s' already exists in database. Delete downloaded file and index it for
current user.", hash, userID));
            }
            // delete downloaded file
            file.delete();

            if (result != null) {
                // new relation is created. Index file for new user
                new IndexThread(existingFile.getParsedLocation(), userFileRel.getID(), name, userID, hash).start();
            }
        } else {
            // file doesn't exists - process it further

            // get user to retrieve synthesizer settings
            UserItem user = UserManager.getInstance().getUser(userID);

            // parse the file content
            File parsedFile = new File(mParsedFilesDir, hash);
            if (sLog.isDebugEnabled()) {
                sLog.debug(String.format("Parsing file content and store it in '%s'.",  parsedFile.getCanonicalPath()));
            }
            File speechBySlidesLocation = null;
            ContentMetadata metadata = extractFileContent(file.getCanonicalPath(), parsedFile.getCanonicalPath());

            // check what language to use
            if (user.getSpeechSettings().getLanguage() != LanguageToUse.DETECT) {
                sLog.info("Speech synthesizer forsed to use language: " + user.getSpeechSettings().getLanguage());
              metadata.setLanguage( LanguageDetected.valueOf(user.getSpeechSettings().getLanguage().toString()));
            }

            if (PPTParser.PRESENTATION_CONTENT_TYPES.contains(metadata.getContentType())) {
```

```
                // this is ppt file - process as such
                speechBySlidesLocation = new File(mOriginalFilesDir, hash + ".slides");
                if (sLog.isDebugEnabled()) {
                    sLog.debug(String.format("Given file is presentation type. Store slides information in '%s'.",
speechBySlidesLocation.getCanonicalPath()));
                }
                InputStream is = null;
                BufferedWriter slidesToSpeechOut = null;
                try {
                    is = new FileInputStream(file.getCanonicalPath());
                    PPTParseResult pptParseResult = ParserProvider.getDefaultPPTParser().parse(is,
metadata.getContentType());
                    slidesToSpeechOut = new BufferedWriter(new FileWriter(speechBySlidesLocation));

                    if (pptParseResult.getSlidesImagesBase64Encoded().size() != pptParseResult.getSlidesText().size()) {
                        sLog.error(String.format("Num slide texts %s and images %s differ!",
pptParseResult.getSlidesText().size(), pptParseResult.getSlidesImagesBase64Encoded().size()));
                    } else {
                        // generate speech for slide texts
                        File tmpTextFile = new File(mSpeechFilesDir, hash + ".txt");
                        File tmpSpeechFile = new File(mSpeechFilesDir, hash);
                        for (int i = 0; i < pptParseResult.getSlidesText().size(); i++) {
                            // write text to file, as under windows cmd doesn't support cyrilic characters
                            Writer fileWriter = null;
                            try {
                                fileWriter = Files.newBufferedWriter(Paths.get(tmpTextFile.getCanonicalPath()),
StandardCharsets.UTF_8);
                                fileWriter.write(pptParseResult.getSlidesText().get(i));
                                fileWriter.flush();
                            } catch (IOException ex) {
                                sLog.error(ex.getMessage(), ex);
                            } finally {

                                if (fileWriter != null) {
                                    try {
                                        fileWriter.close();
                                    } catch (IOException e) {
                                    }
                                }
```

```
                }


                if (sLog.isDebugEnabled()) {
                    sLog.debug(String.format("Generating speech for slide with content '%s'.",
pptParseResult.getSlidesText().get(i)));
                }
               TextSynthesizerProvider.getDefaultSynthesizer(
SynthesizerLanguage.valueOf( metadata.getLanguage().toString()),
SpeechSettingsHelper.getSynthesizerSettings(  TextSynthesizerProvider.getSynthesizerType(),
user.getSpeechSettings())).synthesizeFromFileToFile(  tmpTextFile.getCanonicalPath(),
tmpSpeechFile.getCanonicalPath());


                // transfer bytes
                ByteArrayOutputStream tmpOut = null;
                FileInputStream tmpIn = null;
                try {
                    tmpOut = new ByteArrayOutputStream();
                    tmpIn = new FileInputStream(tmpSpeechFile);
                    IOUtils.copy(tmpIn, tmpOut);
                } catch (IOException e) {
                    sLog.error(e.getMessage(), e);
                } finally {
                    if (tmpIn != null) {
                        try {
                            tmpIn.close();
                        } catch (IOException e) {
                        }
                    }
                    if (tmpOut != null) {
                        try {
                            tmpOut.close();
                        } catch (IOException e) {
                        }
                    }
                }


                if (i != 0) {
                    slidesToSpeechOut.write(BASE64_SEPARATOR);
                }
```

```
                slidesToSpeechOut.write(SLIDE_SPEECHIMG_START);
                slidesToSpeechOut.write(Base64.encode(tmpOut.toByteArray(), false));
                slidesToSpeechOut.write(BASE64_SEPARATOR);
                slidesToSpeechOut.write(SLIDE_IMG_START);
                slidesToSpeechOut.write(pptParseResult.getSlidesImagesBase64Encoded().get(i));
            }
            if (!tmpSpeechFile.delete()) {
                sLog.error(String.format("Unable to delete temporary hash file '%s'!",
                        tmpSpeechFile.getCanonicalPath()));
            }
            if (!tmpTextFile.delete()) {
                sLog.error(String.format("Unable to delete temporary text file '%s'!",
                        tmpTextFile.getCanonicalPath()));
            }
        }
    } catch (Exception e) {
        sLog.error(e.getMessage(), e);
        result = null;
    } finally {
        if (is != null) {
            try {
                is.close();
            } catch (IOException e) {
            }
        }
        if (slidesToSpeechOut != null) {
            try {
                slidesToSpeechOut.close();
            } catch (IOException e) {
            }
        }
    }
}

// based on language detected - generate speech
File speechFile = new File(mSpeechFilesDir, hash);
if (sLog.isDebugEnabled()) {
```

```
                sLog.debug(String.format( "Generating speech for file '%s' using language '%s' and store it in '%s'.",
                        parsedFile.getCanonicalPath(),  SynthesizerLanguage.valueOf(metadata.getLanguage().toString()),
                        speechFile.getCanonicalPath()));
            }
            TextSynthesizerProvider.getDefaultSynthesizer(
                    SynthesizerLanguage.valueOf(metadata.getLanguage().toString()),
                    SpeechSettingsHelper.getSynthesizerSettings(TextSynthesizerProvider.getSynthesizerType(),
                            user.getSpeechSettings())).synthesizeFromFileToFile(parsedFile.getCanonicalPath(),
                    speechFile.getCanonicalPath());

            // update db record
            FileItem exitingFile = mFilesTable.get(hash);
            if (exitingFile == null) {
                sLog.error(String.format("File with hash %s doesn't exists. Unable to update speech location", hash));
            } else {
                exitingFile.setSpeechLocation(speechFile.getCanonicalPath());
                if (speechBySlidesLocation != null) {
                    exitingFile.setSpeechBySlidesLocation(speechBySlidesLocation.getCanonicalPath());
                }
                exitingFile.setParsedLocation(parsedFile.getCanonicalPath());
                mFilesTable.add(exitingFile);
            }

            // index created file - asynchronously
            new IndexThread(parsedFile.getCanonicalPath(), userFileRel.getID(), name, userID, hash).start();
        }
    }
    return result;
}
```

Приложение 2: Скрипт за пакетиране на финалната програма

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project basedir="./" default="dist" name="UltimateSpeaker">
            <!--ANT 1.7 is required                                        -->
            <dirname property="basedir" file="${ant.file.UltimateSpeaker}" />
            <property name="distdir" value="${basedir}/Deploy" />
```

```
<property name="libsdir" value="${distdir}/libs" />
<property name="jarsdir" value="${distdir}/jars" />
<property name="releasedir" value="${distdir}/release" />
<property name="mainClass" value="com.aasenov.MainClass" />

<!--***  Dependent projects declare section ****-->
<property name="AdminUI.dir" value="AdminUI" />
<property name="AdminUI.buildfile" value="${AdminUI.dir}/build.xml" />
<import file="${AdminUI.buildfile}" optional="true" />

<property name="Data.dir" value="Data" />
<property name="Data.buildfile" value="${Data.dir}/build.xml" />
<import file="${Data.buildfile}" optional="true" />

<property name="Commons.dir" value="Commons" />
<property name="Commons.buildfile" value="${Commons.dir}/build.xml" />
<import file="${Commons.buildfile}" optional="true" />

<property name="RestAPI.dir" value="RestAPI" />
<property name="RestAPI.buildfile" value="${RestAPI.dir}/build.xml" />
<import file="${RestAPI.buildfile}" optional="true" />

<property name="SearchEngine.dir" value="SearchEngine" />
<property name="SearchEngine.buildfile" value="${SearchEngine.dir}/build.xml" />
<import file="${SearchEngine.buildfile}" optional="true" />

<property name="TextParse.dir" value="TextParse" />
<property name="TextParse.buildfile" value="${TextParse.dir}/build.xml" />
<import file="${TextParse.buildfile}" optional="true" />

<property name="TextSynthesis.dir" value="TextSynthesis" />
<property name="TextSynthesis.buildfile" value="${TextSynthesis.dir}/build.xml" />
<import file="${TextSynthesis.buildfile}" optional="true" />

<property name="RestletWadlExt.dir" value="RestletWadlExt" />
<property name="RestletWadlExt.buildfile" value="${RestletWadlExt.dir}/build.xml" />
<import file="${RestletWadlExt.buildfile}" optional="true" />
```

```
<!--******** Dependent projects targets section ********-->

<target depends="RestAPI, AdminUI.dist" name="AdminUI"/>
<target depends="Commons, Data.dist" name="Data"/>
<target depends="Commons.dist" name="Commons"/>
<target depends="Data, SearchEngine, TextParse, TextSynthesis, RestletWadlExt, RestAPI.dist" name="RestAPI"/>
<target depends="Commons, SearchEngine.dist" name="SearchEngine"/>
<target depends="TextParse.dist" name="TextParse"/>
<target depends="TextSynthesis.dist" name="TextSynthesis"/>
<target depends="RestletWadlExt.dist" name="RestletWadlExt"/>



<!--********** Generate classpath section ***********-->
<!-- Include all dependency jars from libs folder. -->
<path id="classpath">
        <fileset dir="${Commons.dir}/lib">
                <include name="**/*.jar" />
        </fileset>
</path>
<!-- convert classpath to a flat list/string for use in manifest task -->
<pathconvert property="mf.classpath" pathsep=" ">
        <path refid="classpath" />
        <flattenmapper />
</pathconvert>

<!--************* Main targets section *************-->
<target name="build" depends="init, AdminUI">
        <!--Copy resources to dist -->
        <copy todir="${releasedir}">
                <fileset dir="${basedir}/AdminUI/src/main/resources">
                        <include name="*" />
                </fileset>
        </copy>

        <!--Copy WebUI dist -->
        <mkdir dir="${releasedir}/UI" />
```

```
		<copy todir="${releasedir}/UI">
			<fileset dir="${basedir}/WebUI/src/main/resources/UltimateSpeaker" />
		</copy>

		<!-- Prepare external libraries -->
		<copy todir="${libsdir}" flatten="true">
			<fileset dir="${Commons.dir}/lib">
				<include name="*.jar" />
			</fileset>
			<fileset dir="${Commons.dir}/lib">
				<include name="**/*.jar" />
			</fileset>
		</copy>
	</target>

	<target name="clean">
		<echo>UltimateSpeaker : removing build directories</echo>
		<delete dir="${distdir}" />
	</target>

	<!-- init target - Create the build and dist dirs folder structures -->
	<target name="init" depends="clean">
		<echo>UltimateSpeaker : creating build infrastructure</echo>
		<tstamp />
		<mkdir dir="${libsdir}" />
		<mkdir dir="${jarsdir}" />
		<mkdir dir="${releasedir}" />
	</target>

	<target name="dist" depends="build">
		<jar destfile="${releasedir}/UltimateSpeaker.jar">
			<manifest>
				<attribute name="Main-Class"
value="org.eclipse.jdt.internal.jarinjarloader.JarRsrcLoader" />
				<attribute name="Rsrc-Main-Class" value="${mainClass}" />
				<attribute name="Class-Path" value="." />
				<attribute name="Rsrc-Class-Path" value="./ ${mf.classpath}" />
			</manifest>
```

```
                <zipfileset src="jar-in-jar-loader.zip" />
                <zipgroupfileset dir="${jarsdir}" includes="*.jar"/>
                <zipfileset dir="${libsdir}/" includes="*.jar" />
            </jar>
        </target>
</project>
```

Приложение 3: Страница за показване на онлайн презентации

```
<!DOCTYPE HTML>
<html lang="bg">
<head>
  <meta charset="UTF-8">
  <title>Ultimate Speaker</title>
  <link rel="stylesheet" href="scripts/reveal.css">
  <link rel="stylesheet" href="scripts/theme/black.css" id="theme">
  <script src="scripts/reveal.js"></script>
  <script src="scripts/jquery.min.js"></script>
  <script>
  var autoSound = false;
  var messageReceived = false;

  function loadSlides(fileSlides){
    var htmlToDisplay = '<div class="reveal">';
    htmlToDisplay += '<div class="slides">';
    for(i=0; i<fileSlides.Image.length ; i++){
      htmlToDisplay += '<section class="oneSlide">';
      htmlToDisplay += '<img class="soundControlOff" src="scripts/images/SoundOff.png"
onClick="toogleAutoSpeech(this);" style="width:5%; height:5%; border: 0; background: none; margin: 0; position
absolute; float: right; cursor:pointer;"  alt="Auto speek control button" title="Toogle auto speech" />';
      htmlToDisplay += '<img class="soundControlOn" src="scripts/images/SoundOn.png"
onClick="toogleAutoSpeech(this);" style="width:5%; height:5%; border: 0; background: none; margin: 0; position
absolute; float: right; cursor:pointer; display: none;"  alt="Auto speek control button" title="Toogle auto speech" />';
      htmlToDisplay += '<img src="' + fileSlides.Image[i] + '" alt="Slide' + (i+1) + '" />';
      htmlToDisplay += '<audio controls>';
      htmlToDisplay += '<source src="data:audio/wav;base64,' + fileSlides.Speech[i] + '" />';
      htmlToDisplay += '</audio>';
      htmlToDisplay += '</section>';
```

```
    }
    htmlToDisplay += '</div>';
    htmlToDisplay += '</div>';

    $("body").html(htmlToDisplay)

    initReveal()
  }

  function playCurrentSlide() {
   Array.prototype.slice.call( document.querySelectorAll( ".oneSlide" ) ).forEach( function( slide ) {
    var audio = slide.querySelector( "audio" );
    if( audio ) {
      if( autoSound && slide.classList.contains( "present" ) ) {
        if(audio.currentTime != 0){
          audio.currentTime=0;
        }
        audio.play();
      } else {
        audio.pause();
      }
    }
   });
  }

  function toogleAutoSpeech(button){
    if(button.getAttribute("src").indexOf("SoundOn.png") > -1){
      autoSound = false;
      Array.prototype.slice.call(document.querySelectorAll(".soundControlOn")).forEach( function( button )
{button.style.display = 'none';});
      Array.prototype.slice.call(document.querySelectorAll(".soundControlOff")).forEach( function( button )
{button.style.display = '';});
    } else {
      autoSound = true;
      Array.prototype.slice.call(document.querySelectorAll(".soundControlOn")).forEach( function( button )
{button.style.display = '';});
      Array.prototype.slice.call(document.querySelectorAll(".soundControlOff")).forEach( function( button )
{button.style.display = 'none';});
    }
```

```
  playCurrentSlide();
}

function initReveal(){
  Reveal.initialize({ controls: true, progress: true, slideNumber: true, slideNumber: 'c / t', center: true, transition: "slide"});
  Reveal.addEventListener( "ready", function( event ) {playCurrentSlide();} );
  Reveal.addEventListener( "slidechanged", function( event ) {playCurrentSlide();} );
}

function displayMessage(event){
  if(messageReceived){
    console.log("Message already received from page. Skip processing!");
    return;
  }
  messageReceived = true;
  var settings = event.data;

  //send back the interval ID to be stopped.
  event.source.postMessage(settings.intervalID,"*");

  if(typeof settings.fileID == 'undefined' ){
    $("body").html("<h1 style='color:white;'>FileID attribute is empty! Please open the page again.</h1>");
    $("body").attr("style","");
  } else {
    $.ajax({
    url: settings.serverURL + 'management/files/' + settings.fileID, //file ID is set from parent window
    method: "GET",
    data: {
      type : "slides"
    },
    dataType: 'json',
    beforeSend: function (xhr) {
      xhr.setRequestHeader( 'Authorization', 'Basic ' + btoa(settings.userMail + ':' + settings.userPass));
    },
    success: function(data) {
      loadSlides(data);
      $("body").attr("style","");
```

```
    },
    error: function( data, textStatus, errorThrown ) {
      $("body").html(data.responseText);
      $("body").attr("style","");
    }
    });
  }
}

$(document).ready(function() {
  if($(".reveal").get(0)){
    //page is already loaded. do nothing.
    initReveal()
    $("body").attr("style","");
    return;
  }

  if (window.addEventListener) {
    // For standards-compliant web browsers
    window.addEventListener("message", displayMessage, false);
  } else {
    window.attachEvent("onmessage", displayMessage);
  }
});
 </script>
</head>
<body style="cursor: wait; position: fixed; width:100%; height:100%">
</body>
</html>
```