Interactive Personalized eBooks for Education[*]

Stavros Droutsas[**]
School of Applied Mathematical and Physical Sciences
National Technical U. of Athens
15780 Zografou, Athens, Greece
Email: stavrosdro@gmail.com

Panagiotis Patsilinakos
School of Electrical and Computer Engineering
National Technical U. of Athens
15780 Zografou, Athens, Greece
Email: patsilinak@mail.ntua.gr

Antonios Symvonis
School of Applied Mathematical and Physical Sciences
National Technical U. of Athens
15780 Zografou, Athens, Greece
Email: symvonis@math.ntua.gr

***Abstract*—In the past years, more and more research is carried out in order to incorporate new technologies to modern educational systems. This process does not only provide new means as alternatives to older ones (like printed books) but also fundamentally changes the educational process. Perhaps the biggest advantage new technologies have to offer is the ability to provide a personalized learning experience to each student according to their demands and abilities. In this paper we describe a new design that makes use of the latest eBook standard's (ePub3) features, in order to create a personalized system for learning.**

1. Introduction

Our main goal is to design a system that creates and uses interactive and personalized eBooks, for educational purposes. In this context, by *"interactive"* we mean that there are several *activities* embedded in the eBook that the user can use/play during reading through an eBook reader. By *"personalized"* we mean that each eBook is created for a specific user, in such a way that it's content depends on the user's reading history, performance and personal preferences.

In our work we focus only on eBooks following the new IDPF eBook specification, *ePub3* [7].The ePub3 specification defines a file format used for encoding and representing structured Web content, including HTML5 [11] and CSS. Interactivity is also provided by supporting JavaScript [16]. This means that an eBook with the ePub3 specification, could contain interactive educational activities/games, created with HTML5 and JavaScript, just as in a web-page (See [2] for an example).

There are other standards for electronic documents that provide interactivity. A popular example is interactive Adobe PDF documents. Adobe supports "JavaScript in Adobe Acrobat software", based on JavaScript [15]. Using this software one can create interactive PDF documents with fillable forms, dynamic appearance etc. The main reason this, or other interactive standards, were not considered for this work is that they are proprietary and implemented by few commercial eBook readers. The ePub3 standard is a universal standard for eBook readers (See also [3]).

The described system supports the full cycle of the process of interactive personalized eBook reading, i.e., *the creation* of personalized eBooks, the eBooks' reading/playing functionality and the management of *user reading/playing history* that is utilized when producing a new eBook for a particular user. Creating a new eBook for a particular user may involve user models (which may be external to our eBook system) that capture the user's skills and educational goals and utilize the user's previous performance as indicated by her reading/playing history. In this spirit, in the context of the iRead project, our eBook system can interface with the iRead system [12], in order to provide personalized interactive eBooks which support the process of *learning to read*.

2. User requirements: A typical use-case

Our aim is to support the full life-cycle of interactive personalized eBooks. Briefly, upon request, a service must generate an eBook, tailored to the needs of a specific user, containing interactive educational activities/games relevant to a supported knowledge field. Then, the user must be able to read/play the eBook on her device, save her reading data (actions/playing-history) locally and have these data uploaded to a system server. When closing and later reopening the eBook, the user must be able to continue reading the eBook from the point where she stopped reading. As a typical example, consider an ebook containing crosswords (the interactive activity) where the words in the crosswords have been selected from a list of words the user has to become familiar with (the knowledge field). The user can start solving crosswords in the eBook. The eBook can accept/reject the correct/wrong answers and record the user's gaming activity (number of tries, time needed to find next correct word, etc). It can also provide hints (starting letter, one vowel, etc) as part of its interactivity as well

[*]This work was partially funded by the iRead (project no: 731724) H2020 Innovation Action EU Project.
[**]The work of S. Droutsas was conducted as part of his Diploma Thesis under the supervision of Prof. A. Symvonis

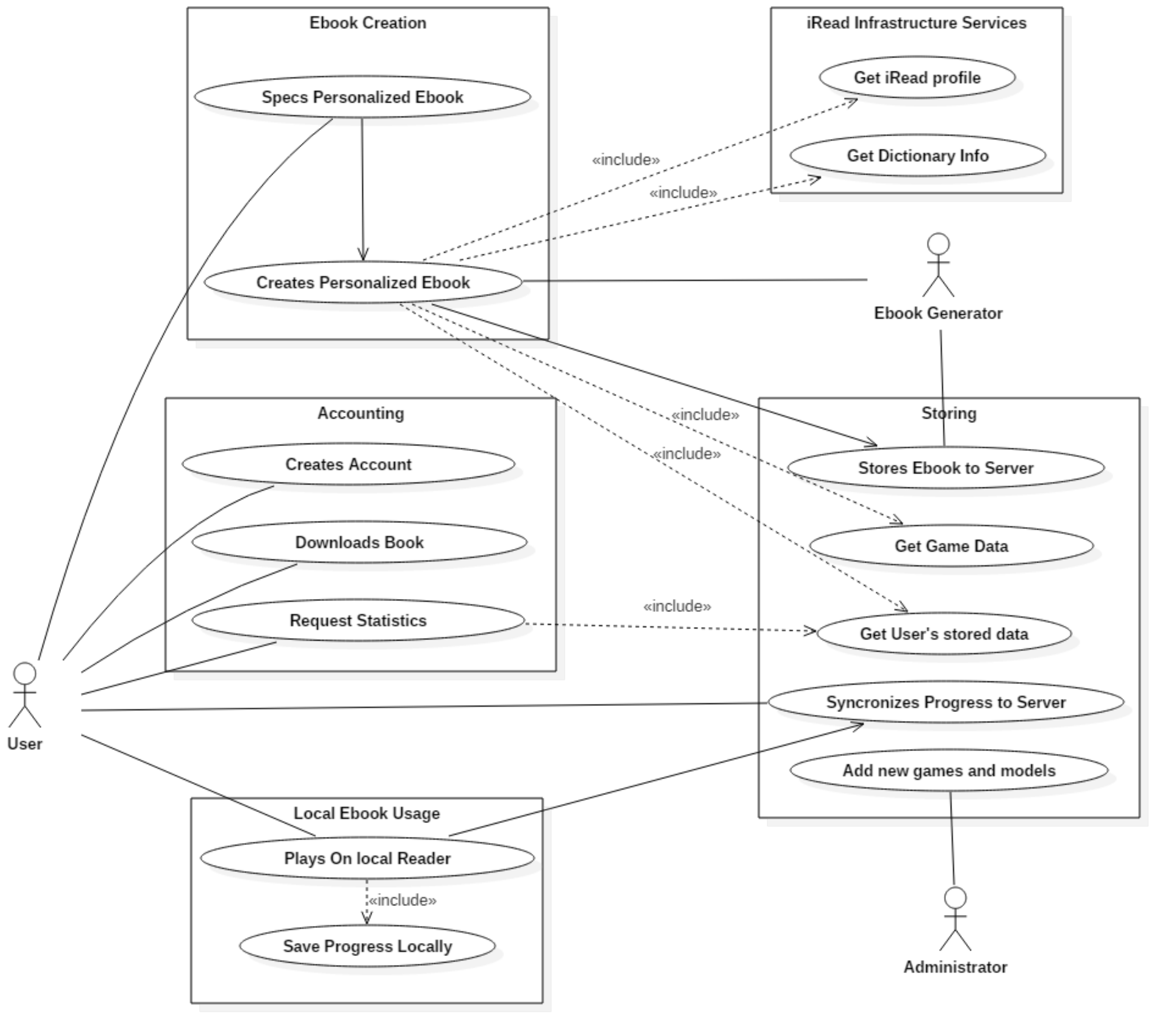


Figure 1. UML use-case description.

as keeping/displaying a score regarding the user's performance/progress. If the user closes the eBook she must be able to resume from where she stopped, that is, all partial progress on the crosswords must be present when the eBook is re-opened. Besides providing a *restore point* for a re-opened eBook, the stored reading/playing history can be utilized for an equally important purpose: the creation of the next eBook for the specific user. Data on previous performance can guide (based on a content generation algorithm) to the creation of the next eBook and, eventually, to the creation and update of a user-model.

The described system, provides the following services and work-flow to a user:

- The user creates an account in our eBook server. For the time being, there is no support for different types of users (e.g., teachers/students).
- The user chooses the type of eBook to be created by selecting one of the available domain models (knowledge domains).
- On the eBook creator application, referred to as *eBook synthesizer*, she specifies the eBook specifics, that is, the number of eBook pages (one activity per page), which activities to include, what is their difficulty level, etc. Of course, default values can be selected on behalf of the user.
- If the user has already read eBooks created by our system for the same model, the eBook synthesizer takes into account the user's stored reading history and playing performance to automatically select appropriate content for the new eBook under creation.
- After the eBook is created, the user downloads it from the server to her tabloid.
- The user reads (plays) the eBook on her favorite ePub3 compliant eBook reader. During play, progress is being locally saved either automatically or manually through controls embedded in the eBook.

- Locally saved data (progress) is synced to the main data server either automatically or manually (again through controls available in the eBook).
- The user can login to the eBook server, to see her overall progress, saved eBooks etc.

Figure 8 provides UML description for the described use-case. The *iRead Infrastructure services* that appear on the top-right part of the figure are responsible for providing specific content when the utilized knowledge model is that developed for the iRead project targeting the skill of "learning how-to-read" (See Section 4).

3. The eBook system architecture

We proceed to describe the system's architecture. In Section 3.1, we describe the architectural and technical choices made for the local playing/reading of the eBook on the user's tabloid. In Section 3.2, we describe an architectural design for the full eBook system. Figure 2 presents a schematic representation of the described architecture.

3.1. Local playing/saving architecture

A major requirement of any eBook system is that the user is able to read the eBooks she has downloaded offline (i.e., when Internet connection is not available). Subsequently, in order for the system to be functional, a user reading/playing an eBook must be able to locally save her progress and continue playing from where she stopped at any point in the future. Moreover, we want this locally saved information to be synchronized with our remote server which keeps track of the user's progress/playing-history so that our system can utilize this information for the production of the next eBook devoted to the specific user.

The main problem that we must overcome, is that the ePub3 standard does not specify a saving functionality that ePub3 compliant eBook readers must implement. There are some eBook readers that provide some saving functions (e.g., see gitden reader [10], AZARDI reader [4]). Each of them though, has a custom, reader specific, saving mechanism. Alternatively, we could create a wrapper application for some popular eBook reader in order to both provide local save and remote sync capabilities for an eBook reader that doesn't have local storage.

Both of the above solutions suffer from the fact that only one (or very few) reader can be used by the end user in order to play eBooks created by our application. This would considerably limit the usability of our system since we want to allow the user to select any commercial eBook reader that implements the ePub3 standard.

In order to overcome the above problem, we chose to create a server application that runs on the background of user's tabloid (**local save/sync server**). The purpose of the local save/sync server is to implement and support all the required save and sync functionality, on the user's tabloid. Given that, the user only has to install our "helper" application (which packages the local save/sync server) on her device and then use the created interactive eBooks through her favorite pre-installed eBook reader.

Keep in mind, that in this design, or an equivalent, it wouldn't be necessary for the user's tabloid to always be on-line.

3.1.1. Local playing/saving architecture specifics. Initially, an eBook is created for some specific user. Each eBook contains a unique serial number that is associated to this specific user in our remote server. The user then downloads the eBook on her device and reads it.

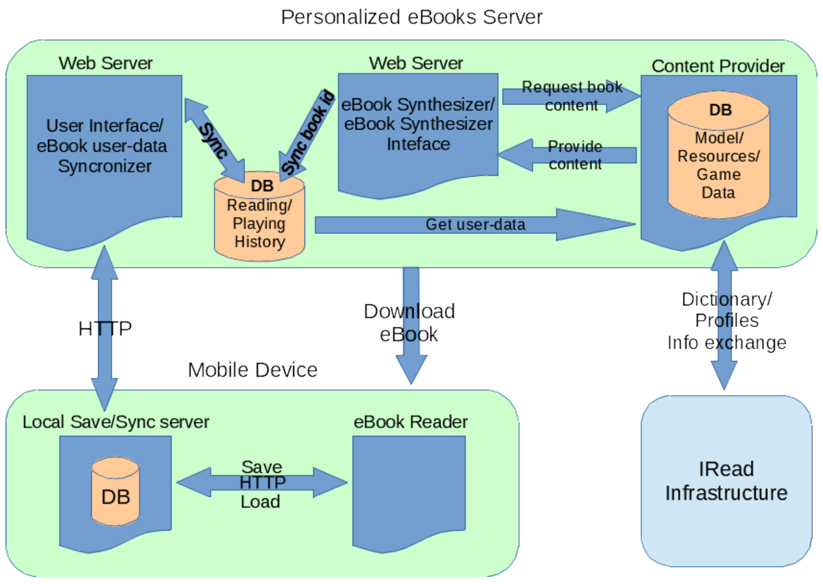
Since we want the local save/sync server to work with any eBook reader that implements the ePub3 standard, the HTTP protocol is the only protocol that can be considered for the communication between the eBook reader and our local save/sync server application. That is, since HTML5 provides methods for an HTTP request-response between client and server, we can assume that an eBook reader fully implementing HTML5 also implements these methods. No more assumptions can be made though, about the communication protocols supported by the eBook reader. For that reason, our save/sync server application functions as a light web server, listening on some pre-selected port on the user's device to which the eBook reader, through embedded code in the eBook, sends requests.

All data from eBooks used on a particular device are stored using this mechanism in the local save/sync server's local database. This can be done either automatically by code embedded in the eBook or by controls/buttons embedded in the eBook and activated by the user. An eBook can also use the same mechanism (HTTP request) to retrieve saved data from the local save/sync server. Using JavaScript embedded in the games source code it can set the game at the previous saved state.

Finally, the save/sync server application can synchronize it's local database data with our remote server's database, containing the progress of all users (the server knows to which user the eBook belongs through the eBook's id). This can also be done automatically or by controls available through the eBook.

3.1.2. Specifications. Any eBook reader fully implementing the ePub3 standard should work with the above architecture. That being said, we must highlight the following key features, that are sometimes missing even from eBook readers that claim to fully implement the standard.

- jQuery [17] support. JQuery is a "cross-platform JavaScript library designed to simplify the client-side scripting of HTML". Even though JavaScript is part of the standard, many eBook readers that are able to support it do not support jQuery. This feature is extremely important for the design of the games/activities.
- The eBook reader must allow cross-domain XHR (xml HTTP requests) or support CORS [5] (cross-origin resource sharing). This is crucial for the communication between the eBook reader and the



Personalized eBooks Server
Web Server
User Interface/ eBook user-data Syncronizer
Sync
Sync book id
DB Reading/ Playing History
Web Server
eBook Synthesizer/ eBook Synthesizer Inteface
Request book content
Provide content
Content Provider
DB Model/ Resources/ Game Data
Get user-data
HTTP
Download eBook
Dictionary/ Profiles Info exchange
Mobile Device
Local Save/Sync server
DB
Save HTTP Load
eBook Reader
IRead Infrastructure

Figure 2. The architecture of our interactive personalized eBook system.

save/sync server application which runs on the user's tabloid, which is impossible otherwise.

- Despite the fact that it is not part of the specifications or necessary to our architectures, it is useful for the design of the games if the eBook reader allows to disable page changing by gestures. For example touch gestures within a specific HTML element (e.g., gestures used in a game) could be excluded from eBooks reader page changing controls in order to avoid confusion with the game's functionality.

The following commercial eBook readers have been tested (only in Android) and found to support the proposed architecture:

- ePub Reader for Android [8]
- GitDen Reader [10]
- Kotobee Reader [18]
- Supreader [6]

3.2. System architecture

Having defined the required client-side architecture, we proceed to describe the full system's architecture. The following is a list of the required services and systems (also refer to Fig. 2).

Server-side (Personalized eBook Server):

- User interface web service (for account creation, account management access to services and data etc.)
- eBook synthesizer interface (web service).
- eBook synthesizer.
- Content provider.
- eBook user-data synchronizer system (proposed to be a web service).
- User-data database (containing eBooks per user, playing/reading history and user-account info).
- Games and model database (containing info regarding the structure of games and user-models used in content creation).

Client-side (Mobile device, tabloid):

- eBook reader (commercial).
- Local save/sync server.
- Local user-data database (contained on local save/sync server).

For the server-side services, there is no technological restriction as to how should they be divided in physical servers. For simplicity in the communication between services, the current design places them all on the same server.

The **eBook synthesizer interface** is the service the user uses to design the eBook she wants. The user has options regarding the morphology of the book (such as number of pages, games per page, etc), and the content. Content-wise, the user has options relating to the topic of the eBook (i.e., the user model the eBook is based on), the included activities/games, the difficulty level, etc. In addition, the user may have the option to specify/select the eBooks content manually or to request it to be generated automatically based on the user's model and the user's playing/reading history. During this process the eBook synthesizer interface communicates with the *content provider* to retrieve info about the available topics, resources and related games and consequently informs the user. When the user makes her selections, the eBook synthesizer interface requests related content (according to the user's options) from the content provider.

The **eBook synthesizer system** receives content from the content provider, trough an appropriate API, and mechanically composes the eBook (according to the ePub3 standard). Note that, the (JavaScript) code of the games/activities to be included in the eBook is part of the provided content. Finally, the eBook synthesizer system informs the *user-data database* about the eBook-id of the new eBook, associating it to the user who requested it, and the content included in the eBook.

The **content provider**, is responsible for identifying content to be included in a new eBook. In a local database (*games and model database*), the content provider contains user-models (topics), related resources and related games/activities. Upon request, the content provider selects resources and related games relevant to the selected model and forwards them to the eBook synthesizer system. The automatic content generation by the content provider is typically based on the user's playing/reading history which is retrieved by accessing the *user-data database*.

The **eBook user-data synchronizer system** is the system with witch the **local save/sync server** on the user's tabloid communicates to sync local progress to the remote server. Note that the **local save/sync server** could directly communicate to a remote database. The above setup was selected for consistency and security reasons (the **eBook progress synchronizer system** verifies consistency and the databases are not publicly accessible).

The *user interface web service*, *eBook synthesizer interface*, *eBook synthesizer system* and *eBook user-data synchronizer system* could be all provided by the same web server.

3.2.1. Communication between services.

Communication that takes place between the system components:

1) **Communication between the commercial eBook reader and the local save/sync server:** As mentioned above, we utilize the HTTP communication protocol. For this reason, the use of the lightweight NanoHTTPD java web server [19] (under a modified BSD license) is proposed for the local save/sync server.
2) **Communication between the local save/sync server and the eBook user-data synchronizer system:** For ease of communication and server-side management, it is proposed that the eBook user-data synchronizer system should be a web service which, in turn, allows the two systems to communicate via the HTTP protocol.
3) **Communication of the content provider and external user-modeling infrastructure:** The content provider can utilize an external source in order to receive user modeling services and relevant resources. In Fig. 2 the iRead project [12] plays the role of the external modeling infrastructure. The content provider should be able to get a user's profile information after redirecting the user to the external user verification sub-system. The content provider should also be able to access other external services and resources.

4. Interacting with external modeling service: the iRead case.

As described in Section 3.2, the *Content Provider* is responsible for the logic and the resources used in developing a new eBook for a specific user. Typically, the Content Provider has to maintain a user model for each specific user and to update it based on the user's reading/playing history. As it is indicated in Fig. 2, the iRead infrastructure can play the role of an external modeling service. In this Section, we briefly introduce the iRead modeling infrastructure and we describe how our personalized eBook system can interface with it.

4.1. The iRead project

iRead is the acronym for the *"Infrastructure and integrated tools for personalized learning of reading skills"*, European Union Horizon-2020 funded project [12]. iRead is a 4-year (2017-2020) project that aims to develop personalized learning technologies to support reading skills. The iRead software combines a diverse set of personalized learning applications and teaching tools for formative assessment. iRead focuses on primary school children across Europe, learning to read and learning English as a foreign language including children with dyslexia who are at risk of exclusion from their education system. The project comprises 15 partners from across industry and education in 8 European countries.

iRead targets the learning skill of reading, covering several languages (English, German, Greek, Spanish). In the context of knowledge representation, we construct (based on expert knowledge) a *domain model* consisting of several (a few hundreds for the studied languages) *features*. The features represent the knowledge that has to be mastered in order to acquire a skill (reading in the case of iRead). The features are linked together so that the notion of *prerequisites* is encapsulated in iRead's models, reflecting the natural way in which language features are mastered/acquired and taught. The features in iRead's domain models are grouped into several categories, including *phonology, morphology, morphosyntax* and *syntax* [13]. The domain models are internally represented in iRead as a *weighted directed acyclic graph*. Node weights represent the level of mastery of each specific feature while edge weights are used for "opening" features for practice. For more details on iRead's architecture refer to [14].

For each child/user, the iRead system creates a *user-model*, also referred to as *user profile*, based on the domain model of the targeted language. In a sense, the user-model is an instantiation of the domain-model where the node weights reflect the level of the reading skill of the specific user. The user model is initialized with either a default profile (e.g., low-level reader) or with values determined by a "reading test" taken by the child. During the course of learning, the node weights increase and, as a result, now features open and become available for practice/learning.

Eventually, the user reaches a high level of competence in all domain-model features, in effect mastering the targeted skill.

So far, we covered the basic ideas of iRead's domain and user modeling. One aspect of the system that we did not mention concerns the way iRead promotes learning. iRead employs linguistic smart games that target the features of the domain model. A large number of linguistic games has been developed (currently about ten games) in order to cover the large number of domain features with relevant activities. While a child plays, success in the game indicates progress in the acquisition of a specific skill. This is reflected in the user model by increasing the user competence for the relevant feature (i.e., increasing the corresponding node weight in the user's model). Of course, as learning to read is a slow process, typically mastered in more than one school year, the evolution of the user's model from "naive-reader" to "expert-reader" takes time. The rules of the evolution take into account the child's playing history and use it as evidence for the improvement of a skill/feature. When "enough" evidence becomes available, the feature's node-weight is increased, moving the user to a new level of mastery for the language domain model in question.

From the above discussion, it is evident that the iRead system can make use of our personalized eBook system in order to deliver learning material to a child in the form of linguistic activities. These activities should utilize content available for a specific language (domain model, educational topic) and for a specific user (as indicated by her user-model). In order to avoid duplicating the user-models, it makes sense to interface with iRead and build eBooks based on user-models already developed in the context of iRead.

4.2. Connecting to iRead

The iRead project aims to develop personalized learning technologies to support reading skills. As part of the iRead project, we have identified the *personalized eBook generation* as a potential learning technology relevant to our modeling infrastructure. So, we specify a way for our eBook system to interface with iRead and utilize its models and resources.

Consider a user who has an iRead account and also wants to order a personalized eBook for the language she is learning to read. During the eBook creation, the synthesizer interface will redirect the user to an iRead log-in page, retrieve the data required (her user-model and relevant content) and create the eBook. After the completion of this process no data of the user's iRead profile are stored.

For the implementation of this functionality, we will use the iRead infrastructure's API, which provides the following features:

- Return a user-model, after authentication with the user's credentials.
- Return linguistic resources relevant to the domain model and appropriate for the user. These resources will be used in the activities included in the eBook.

5. Design challenges

The main issue that arises (independently of the selected architecture) is that of data consistency. Specifically, the question arises when a user reads the same eBook on different devices. Since the user must be able to locally save her progress regardless of whether her device is on-line, there could be multiple, inconsistent, instances of saved data for the same eBook on different devices. For this reason, an appropriate *synchronization policy* must be developed.

Another issue that deserves investigation is that of data privacy. The designed system should conform with the new *General Data Protection Regulation (GDPR)* [9] coming in effect on May 25, 2018.

6. Implementation

As a proof of concept, we provide an implementation of the two key components of the system, required for local playing and remote saving. Namely, the **Local Save/Sync server** and the **eBook user-data Synchronizer**. Also, a basic version of the **user interface web service** is created, that shows user progress uploaded to the remote server. In order to test this implementation, an interactive "crossword" game was also created and embedded into eBooks.

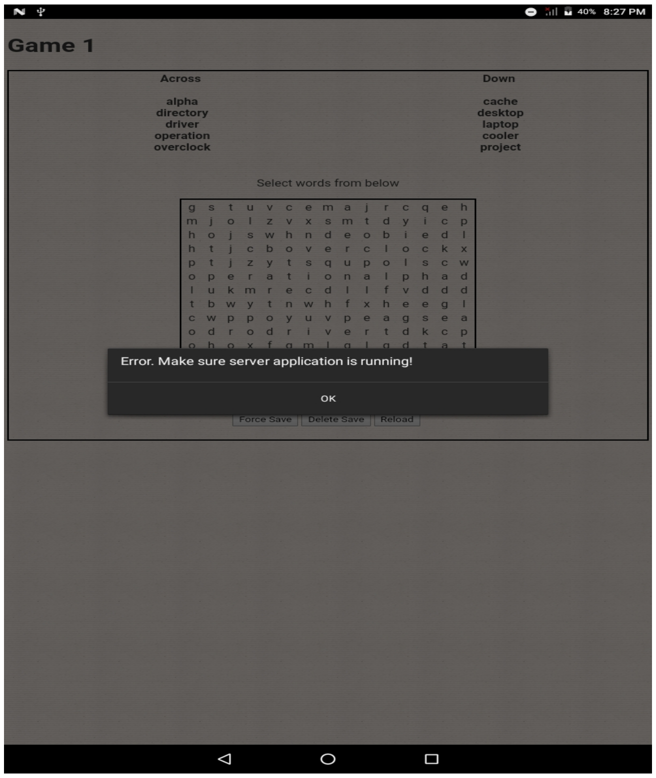
As a test case, we created three eBooks, two for one user, and for another. In each page, of an eBook, a crossword puzzle is embedded, with clues selected from a specific subject. Additionally there are control buttons in each page, that the user can use to save-receive (or delete) data to the local save/sync server. Finally, the first page of each eBook contains a "sync to remote" button, that a user can click in order to make the local save/sync server synchronize local progress to the remote server.

Figures 3-8 are screen-shots of using the provided implementation on an android tablet, using the "ePub Reader for Android" reader. Keep in mind that other than opening the local Save/Sync server in the background, the user only interacts with her favorite ePub reader.

7. Proposed future work

In this implementation, user modeling is kept at a very basic level. Since we would want to use this system for education, it would be useful to create a more robust user modeling that contains the notions of "teacher" and "student" users. The architecture should then support mass creation of personalized eBooks for students by teachers, provide progress feedback for groups of students etc.

Another aspect for development, is to enhance the architecture in order to prevent bad usage. For example, since an eBook is basically a collection of HTML and JavaScript files, a user could easily read and edit information in the eBook used by our architecture, such as bookId. This means that the user could alter the code in the eBook to provide false progress data, or even present data in the name of other users. One possible solution, could be encrypting functional information in the eBook.



Game 1

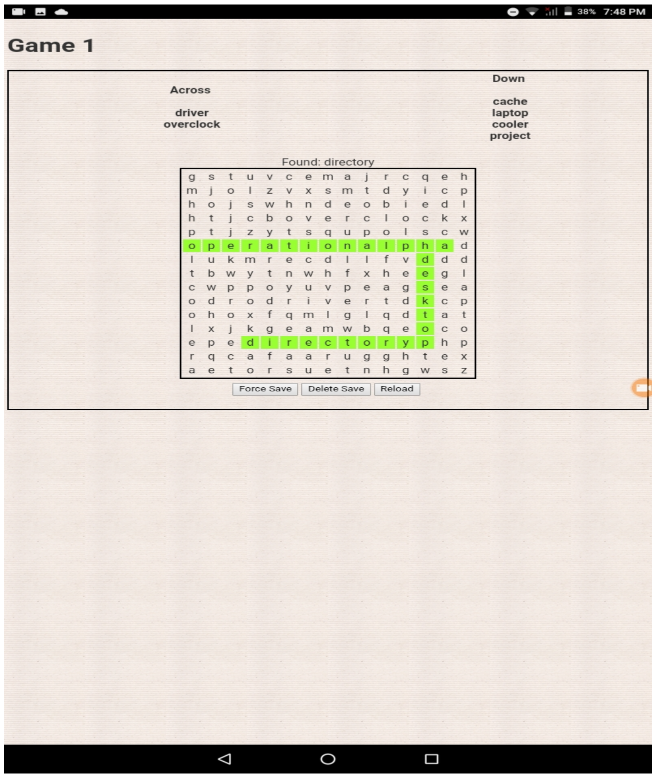
Across: alpha, directory, driver, operation, overclock

Down: cache, desktop, laptop, cooler, project

Select words from below

Error. Make sure server application is running!

Figure 3. User opens game page and is prompted to open the local server app.



Game 1

Across: driver, overclock

Down: cache, laptop, cooler, project

Found: directory

```
g s t u v c e m a j r c q e h
m j o l z v x s m t d y i c p
h o j s w h n d e o b i e d l
h t j c b o v e r c l o c k x
p t j z y t s q u p o l s c w
o p e r a t i o n a l p h a d
l u k m r e c d l l f v d d d
t b w y t n w h f x h e e g l
c w p p o y u v p e a g s e a
o d r o d r i v e r t d k c p
o h o x f q m l g l q d t a t
l x j k g e a m w b q e o c o
e p e d i r e c t o r y p h p
r q c a f a a r u g g h t e x
a e t o r s u e t n h g w s z
```

Figure 5. User reopens the game page, locally stored progress is loaded and she continues playing from previously saved state.

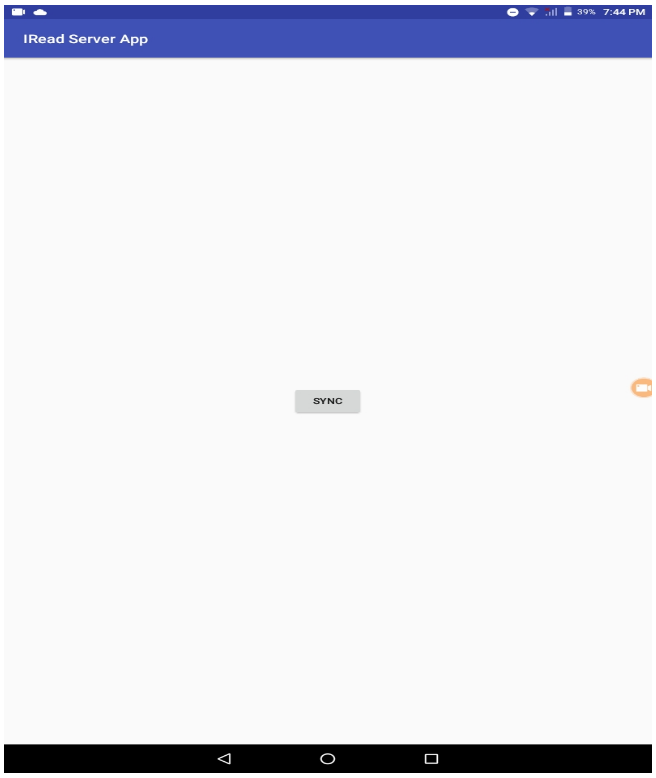
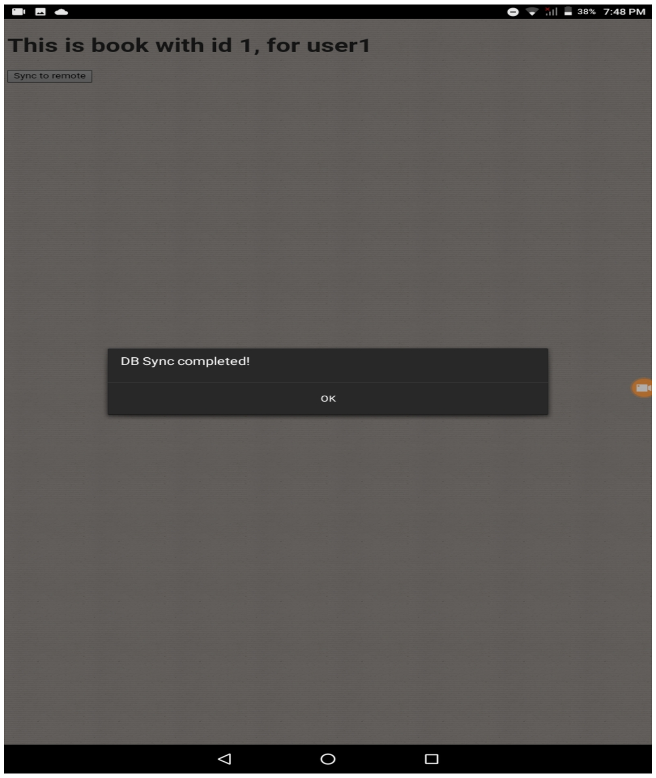


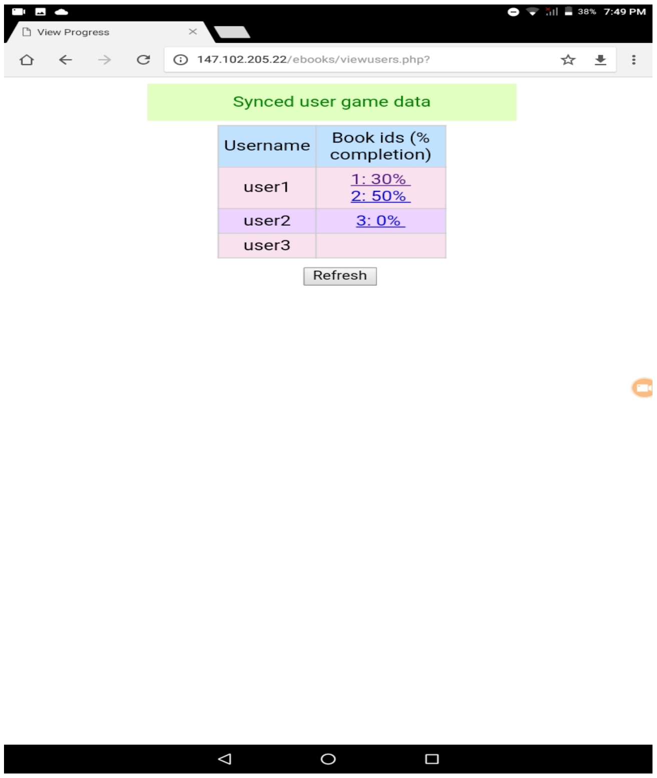
Figure 4. User opens the local Save/Load application and lets it run in the background.



This is book with id 1, for user1

DB Sync completed!

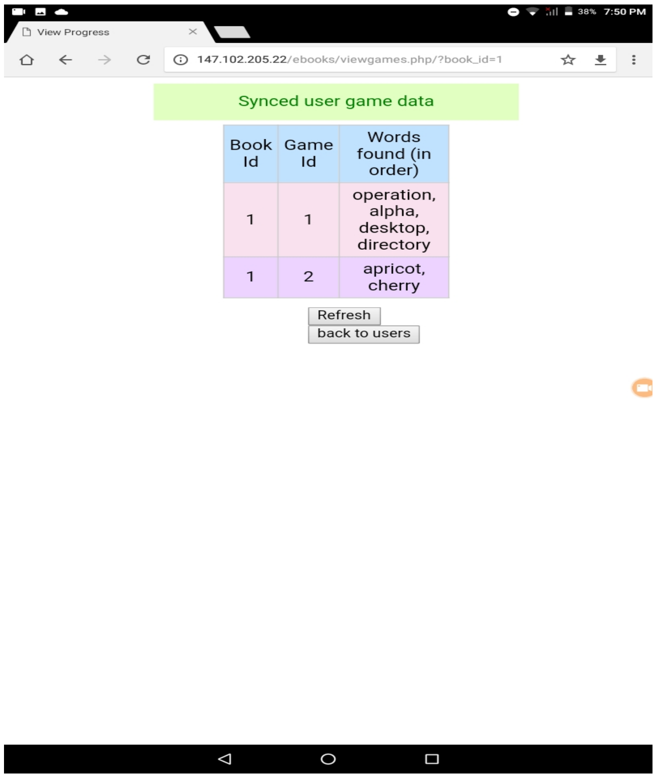
Figure 6. Trough controls in the first page user syncs data to the remote server.



Synced user game data

Username	Book ids (% completion)
user1	1: 30% 2: 50%
user2	3: 0%
user3	

Figure 7. The remote serve associates the book id with the user id to know which progress refers to which user, which book and which game.



Synced user game data

Book Id	Game Id	Words found (in order)
1	1	operation, alpha, desktop, directory
1	2	apricot, cherry

Figure 8. User 1, selects to see her progress in the eBook with id 1.

References

[1] M. Ebner, C. Prettenthaler and M. Hamada, “Cloud-Based Service for eBooks Using EPUB under the Aspect of Learning Analytics,” 2014 IEEE 8th International Symposium on Embedded Multicore/Manycore SoCs, Aizu-Wakamatsu, 2014, pp. 116-122.

[2] S. Okuda and K. Emi, “Make once, play anywhere!: EPUB 3 interactive function enables us to make and play game software anywhere!,” 2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE), Tokyo, 2013, pp. 381-384.

[3] S. Okuda, K. Emi and Y. Kawachi, “Building of an education system with electronic textbooks of the ePub format and with smartphones,” The 1st IEEE Global Conference on Consumer Electronics 2012, Tokyo, 2012, pp. 325-328.

[4] “AZARDI epub3 Reader”, [online] azardi.infogridpacific.com/

[5] “Cross-Origin Resource Sharing (CORS)”, Apr. 2018, [online] https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[6] “ePUB EBook Reader Skoob”, 2017, [online] https://play.google.com/store/apps/details?id=com.s2apps.reader

[7] “Epub3, international digital publishing forum”, 2017, [online] http://idpf.org/epub/30.

[8] “ePub Reader for Android”, 2015, [online] http://www.graphilos.com/epub/Default.aspx

[9] General Data Protection Regulation. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC. Official Journal of the European Union, L 119/1, 4.5.2016.

[10] “Gitden Reader”, 2018, [online] http://gitden.com/gitden-reader/

[11] “HTML5”, Apr. 2018, [online] https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

[12] iRead: Infrastructure and integrated tools for personalized learning of reading skills. European Union Horizon 2020 funded project. [Online] https://iread-project.eu/

[13] iRead deliverable: “D4.1 Dyslexia Domain Models (English and Greek)”. Available online at https://iread-project.eu/ (under publications).

[14] iRead deliverable: “D8.1 System Architecture”. Available online at https://iread-project.eu/ (under publications).

[15] “JavaScript for Acrobat”, Nov. 2006, [online] https://www.adobe.com/devnet/acrobat/javascript.html

[16] “JavaScript, MDN web docs”, 2018, [online] https://developer.mozilla.org/en-US/docs/Web/JavaScript

[17] “jQuery”, 2018, [online] https://jquery.com

[18] “kotobee Reader”, 2018, [online] https://www.kotobee.com/en/products/reader

[19] “NanoHTTPD”, [online] https://github.com/NanoHttpd/nanohttpd