WordSteps Mobile Application

Software Requirements Specification

version: 0.1

Introduction

WordSteps mobile application helps people to learn foreign words using spaced repetition algorithm, which makes this process much more effective so all the words are kept in the memory for a long period of time.

Learning process consists of a number of successive lessons with the optimal interval (in days) between them. Lesson is a set of exercises. Each exercise helps to memorize the word in a specific way: correct reading, correct writing etc.

Optimal intervals between lessons for each word are calculated automatically by spaced repetition algorithm which takes into account the user's current progress e.g. number of correct and incorrect answers in the exercises.

The main feature of the app is to allow users to store all the words and learning stats in the "cloud". The cloud is represented by web-server and could be accessible by HTTP on domain wordsteps.com where every user should have his/her personal account to store the data.

Object Model

There are 2 basic objects in the system: Card and Dictionary.

Card represents single foreign word (or phrase) with its translation to the user's native (or any other understandable) language and with a lot of other parameters to store learning progress. Here is the full list of all the params:

Card *object:*

name	description
phrase_id	ID of the word on the WordSteps server.
dict_id	ID of the dictionary which contains this word
status_id	1 - "learn", 3 - "learned", 4 - "postpone"
phrase	the word (or phrase) itself
translation	translation of the given word (or phrase)
transcription	phonetic transcription of the word. Available only for some of the learning languages.
sound_path	path to the sound file with the voice-over of the word
ease_factor	the value which is calculated by the algorithm (see below). It shows

	how easy this word is for the particular user for the current moment (current lesson).
prev_ease_factor	the value of the *ease_factor* param, calculated for the previous previous lesson.
next_interval	interval in days between the last lesson and the next one for the word.
prev_interval	previous value of the *next_interval* param.
lesson_number	number of lessons passed for the given word.
succ_yes_count	number of successive lessons when the word was answered correctly. If the current answer is incorrect, it turns to 0.
yes_count	number of total lessons when the word was answered correctly.
no_count	number of total lessons when the word was answered incorrectly.
next_lesson_time	UNIX time of the next scheduled lesson.
last_lesson_time	UNIX time of the last passed lesson
first_lesson_time	UNIX time of the first lesson for this word
last_change_time	UNIX time of the last change of any parameter of this card

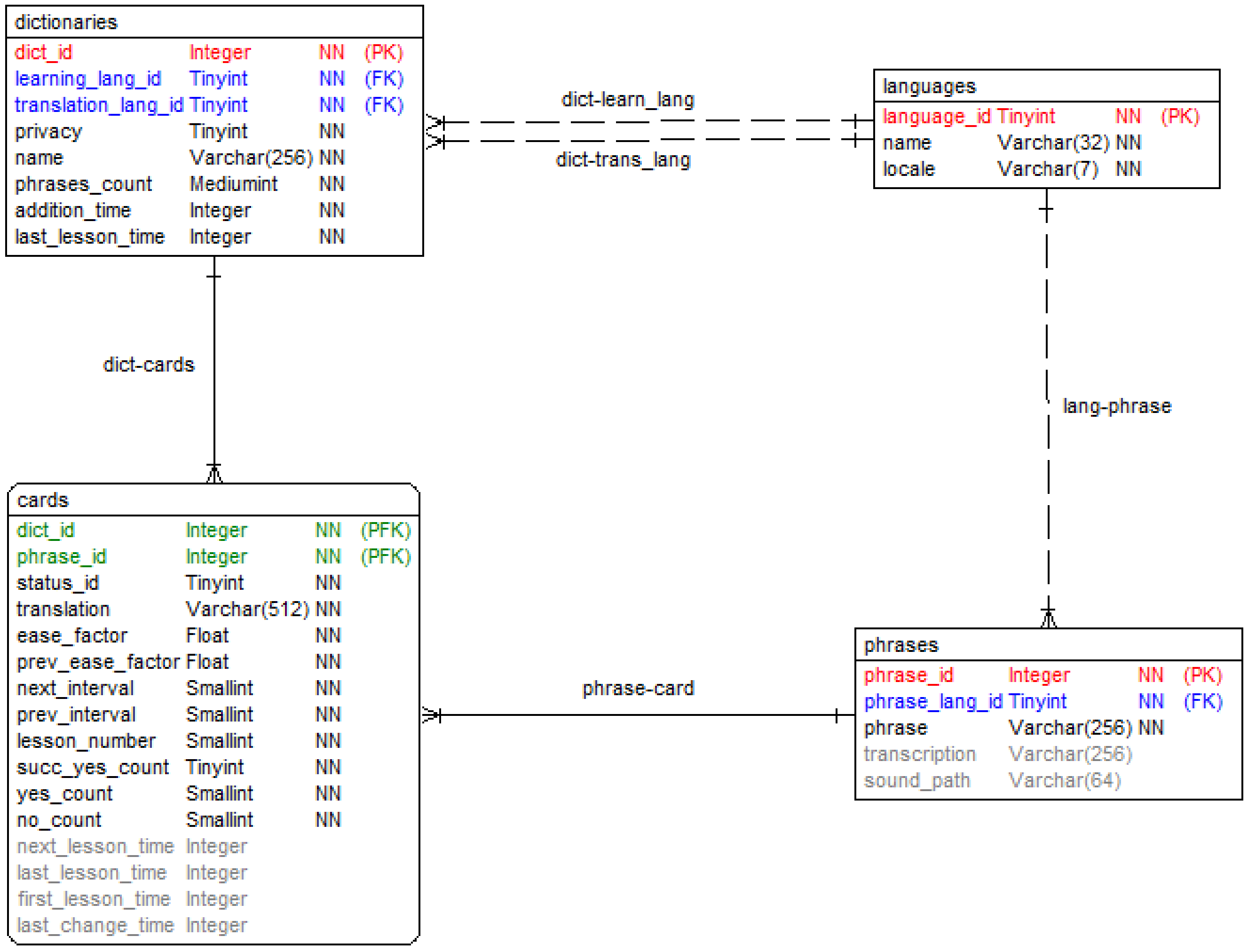
Dictionary contains the list of Cards. It is used to group Cards and to be able to operate with them as with a single set. Maximum number of Cards in the dictionary is 100.

Dictionary *object:*

name	description
dict_id	ID of the dictionary on the WordSteps.com server
learning_lang_id	ID of the learning language of the cards in the dictionary
translation_lang_id	ID of the translation language of the cards in the dictionary
privacy	dictionary privacy: 1 - “public” - available for all on WordSteps.com; 4 - “private” - available only for dictionary owner (creator).
name	name of the dictionary
phrases_count	number of words in the dictionary
addition_time	UNIX time when dictionary was added to the user’s profile (if it is not his/her own dictionary) or created (in case it is his/her own dictionary).

last_lesson_time	UNIX time of the last lesson when at least any of the cards containing in this dictionary are participated in the learning process.

Here is the scratch of WordSteps.com database model which helps to understand the relationships between cards and dictionaries and also contains types of each parameter in terms of MySQL data types. Please take into account that each foreign word could be contained in many dictionaries where it could have different translations and learning statuses. So the word itself should be stored in a separate data table (*phrases* on the pic) from its translations and stats (*cards* on the pic) .



dictionaries			
dict_id	Integer	NN	(PK)
learning_lang_id	Tinyint	NN	(FK)
translation_lang_id	Tinyint	NN	(FK)
privacy	Tinyint	NN	
name	Varchar(256)	NN	
phrases_count	Mediumint	NN	
addition_time	Integer	NN	
last_lesson_time	Integer	NN	

languages			
language_id	Tinyint	NN	(PK)
name	Varchar(32)	NN	
locale	Varchar(7)	NN	

cards			
dict_id	Integer	NN	(PFK)
phrase_id	Integer	NN	(PFK)
status_id	Tinyint	NN	
translation	Varchar(512)	NN	
ease_factor	Float	NN	
prev_ease_factor	Float	NN	
next_interval	Smallint	NN	
prev_interval	Smallint	NN	
lesson_number	Smallint	NN	
succ_yes_count	Tinyint	NN	
yes_count	Smallint	NN	
no_count	Smallint	NN	
next_lesson_time	Integer		
last_lesson_time	Integer		
first_lesson_time	Integer		
last_change_time	Integer		

phrases			
phrase_id	Integer	NN	(PK)
phrase_lang_id	Tinyint	NN	(FK)
phrase	Varchar(256)	NN	
transcription	Varchar(256)		
sound_path	Varchar(64)		

Use Cases

Use cases are grouped by app screens. Take a look at the screens to get a good understanding of the description below.

Screen 1

It is a list of user's dictionaries for the selected learning language. Each user could lean many languages at the same period of time so the screen allows user to switch between them.

Screen also contains statistics for the dictionaries.

Stats by schedule: all the cards in the dictionary could be on any of 3 states according to the schedule, defined by spaced repetition algorithm:

1. *not started* - for the cards which are not yet passed any lessons.
2. *scheduled* - for the cards, which have been learned before and will be ready for the next lessons in the future.
3. *for today* - for the cards, which have been learned before and they are ready for the next lesson right now.

By cards statuses: each card has 3 statuses as mentioned in the object's model description:

- *learn* - cards which participate in the lessons.
- *learned* - cards which already became learned and they are not participated in the lessons.
- *postpone* - cards, manually postponed by the user. They are also not participated in the lessons.

1. Import dictionaries from the server

1. press "Add" button, popup with 2 variants is displayed: "import" and "new"
2. press "import".
3. screen with profile params should be displayed
4. input WordSteps user name and password, press "continue"
5. list of dictionaries to inport is displayed. Select needed ones with checkboxes, press "download"
6. redirect to **screen 1**, where new dictionaries should be displayed.

2. Create new dictionary

1. press "Add" button, popup with 2 variants is displayed: "import" and "new"
2. press "new", ***screen 4*** is displayed.

3. Delete dictionaries

1. select all the necessary dictionaries with the checkboxes.
2. bottom control panel with 2 buttons is displayed: "learn" and "delete" (*iphone_1_selected_dicts_schedule.png*)
3. press "delete": selected dictionaries are deleted from the local dababase including their cards. ***Screen 1*** is displayed and does not contain deleted dictionaries.

4. View progress by schedule

1. select some dictionaries with the ckeckboxes
2. stats panel displays diagram and number of word in selected dictionaries by schedule stats

5. View progress by statuses

1. select some dictionaries with the ckeckboxes
2. stats panel displays diagram and number of word in selected dictionaries by schedule stats (see above)
3. press the arrow at the right of the stats panel. The diagram should change to display stats by cards statuses (see *iphone_1_selected_dicts_statuses.png*). Press again and stats should be changed back to the viewing by schedule.

6. Change current learning language

1. press the learning language flag icon on the top left part of the screen. Drop-down list of available learning languages should be displayed.
2. select another language
3. **Screen 1** is displayed and contained the list of user's dictionaries with cards of the newly selected language (if exist) or empty list.
4. Newly selected learning language should be stored in the system so when **screen 1** is opened again, this language should be displayed by default.
5. Note that the list of available learning languages is presented in **Addendum** section of this document.

7. Start learning process for selected dictionaries

1. select some dictionaries with the ckeckboxes
2. bottom control panel with 2 buttons is displayed: "learn" and "delete" (*iphone_1_selected_dicts_schedule.png*)
3. press "learn". ***Screen 6*** is displayed.

Screen 2

This screen displays foreign words contained in the dictionary. It should contain words, translations, progress bar for each word, phonetic transcription (if supported for current learning language), speaker icons.

1. View progress by schedule

2. View progress by statuses

3. Edit dictionary

1. Press "Edit" button. **Screen 3** should be opened.

4. Start learning process for the dictionary

1. Press "Learn" button. **Screen 6** or **Screen 12** should be opened.

5. Play sound for the words

1. Press to the word field (the whole field, separated by horizontal lines). Sound should be played. If no sound file is available for the word, nothing happens.

Screen 3

This screen allows user to edit current dictionary and its cards.

1. Edit dictionary name

1. Press on the dictionary name. It should be possible to change it. The field should be editable.

2. Add new card

1. Press “New word”.
2. Popup with 2 fields is displayed: “word” and “translation” and 2 buttons: “add” and “close”.
    a. if press “close”, popup is closed.
    b. if press “add”, new word with translation is added to the dictionary, popup window is not closed, offering to input one more word.

3. Delete cards

1. Select cards with checkboxes
2. Press “Delete word” button
3. Cards deleted from the dictionary.

4. Edit cards status

1. Select cards with checkboxes
2. Press “Change status”
3. Popup with drop-down list is displayed. It has 3 options: “learn”, “learned”, “postpone” and “Apply” button.
4. Select needed status and press “Apply”.
5. Popup is closed and new status is successfully applied and displayed for the selected cards.
6. If new status is “learned”, the method *setLearned()* should be called for selected cards (see **Spaced Repetition Algorithm** section).

5. Edit card’s word and translation

1. Press to the card’s field (the field between 2 horizontal lines).
2. Popup with 2 input fields is displayed: “word” with the current word value and “translation” with current translation value. There are also 2 buttons available: “apply” and “cancel”
    a. if “cancel” is pressed, popup is closed
    b. if “apply” is pressed, new values of word and translation are applied to the card, popup is closed.

Screen 4

It displays the form of creation of a new dictionary.

1. Create new dictionary

1. User should input new dictionary name, select learning and translation languages. After pressing “create” button he/she should be redirected to the **Screen 3**.
2. “Learning language” param should be the same as current learning language, displayed on **Screen 1**.
3. If just created dictionary has learning language different from the current one (which should be shored and displayed by default in the **Screen 1**), current learning language should be changed to a new value.
4. If pressing “cancel”, user should be redirected to the **Screen 1**.
5. Note that the list of available learning languages is presented in **Addendum** section of this document.

Screen 5

The screen displays lesson parameters and allows user to change them: number of foreign words in the lesson and sound option.

1. Change number of words in the lesson

1. User can select number of words in the lesson starting from 7 (minimum number) and up to the maximum number of words in selected dictionaries but not more then 100.
2. List should by default contain values 10, 15, 20, 25, 30...max. If selected dictionaries have >= 7 words and <= 10, then the list should contain only this one value.
3. The words that used in the lessons, could only be of “learn” status. Words with other statuses are not taken into account and not counted in this option.
4. When changing number of words during already started lesson, after pressing “save” button popup should be displayed: “To apply this change, the lesson should be restarted” with 2 buttons “ok” and “cancel”. When “ok” is pressed, changes are saved and lesson is restarted with a new number of words to load.

2. Enable/disable sound in the lesson

1. When changing sound option, no lesson restart is needed.
2. If sound is on, it is played in the *Flash Cards* and *Listening Variants* exercises when question is displayed.

Screen 6

Flash Cards exercise: it displays each foreign word in the lesson and its translation.

1. Flash Cards exercise

1. It displays word and translation
2. When new word is displayed, sound with pronunciation is played (in case sound option is enabled)
3. If current learning language has phonetics transcription, it is also displayed.

4. When pressing to the word field, sound should be played again.
5. When pressing “next” button, next word is shown
6. If no more words to display, next exercise is started
7. This exercise is not displayed for the words which are already not under “is being learned” phase (see **Spaced repetition algorithm** section).
8. When “*Settings*” button is pressed, **Screen 5** is displayed. The same is for all other exercises.

Screen 7

Back Flash Cards exercise: displays translation of the word and asks if user remembers the word itself.

1. Back Flash Cards exercise

1. Translation, “Do you know this item?” and yes/no buttons are displayed (*iphone_7_exercise_back_flashcards_ask.png*).
2. When pressing “no”, word is displayed, lesson progress is updated, red error counter is incremented. Then in 3 sec. next word is displayed.
3. When press “yes”, screen *iphone_7_exercise_back_flashcards_ask2.png* is displayed.
    a. if press “correct” - positive green counter is incremented, progress is updated, next word is displayed.
    b. if press “incorrect” - see p.2 of this use case.
4. Erroneous words should be displayed again after all the words have passed this exercise. They should be repeatedly displayed again and again until the user gives correct answer to them.

Screen 8

Words Variants exercise: it displays one translation and suggests user to select correct foreign word for it (5 options).

1. Words Variants exercise

1. Translation and 5 foreign words are displayed, including one correct word.
2. When correct answer is selected, the fields with the answer is highlighted in a green color. After delay in 1 sec. new question is displayed.
3. When incorrect answer is selected, correct field is highlighted in green and selected field - in red (*iphone_8_exercise_words_variants_incorrect.png*). After delay in 3 sec. new question is displayed. During this delay no other answer is selectable.
4. Erroneous words should be displayed again after all the words have passed this exercise. They should be repeatedly displayed again and again until the user gives correct answer to them.

Screen 9

Listening Variants exercise: it suggests to guess correct word translation by its pronunciation. It displays 5 options with one correct translation and 4 incorrect ones (*iphone_9_exercise_listening_variants.png*).

1. Listening Variants exercise

1. When new question is displayed, word pronunciation is played.
2. If no sound file is available for the word, the word itself is displayed (*iphone_9_exercise_listening_variants_no_sound.png*)
3. If sound option id off (see **Screen 5**), the word itself is displayed and there is a sign under the word called “<*sound is off*>”.
4. When correct answer is selected, the fields with the answer is highlighted in a green color. After delay in 1 sec. new question is displayed.
5. When incorrect answer is selected, correct field is highlighted in green and selected field - in red (*iphone_9_exercise_listening_variants_incorrect.png*). After delay in 3 sec. new question is displayed. During this delay no other answer is selectable.
6. Erroneous words should be displayed again after all the words have passed this exercise. They should be repeatedly displayed again and again until the user gives correct answer to them.

Screen 10

Write word exercise: displays translation and suggests user to input correct word.

1. Write word exercise

1. When correct word is input, screen *iphone_10_exercise_write_word.png* is displayed for 1 sec. Then next question is asked.
2. When incorrect word is input, screen *iphone_10_exercise_write_word_incorrect.png* is displayed for 3 sec. Then next question is asked.
3. Erroneous words should be displayed again after all the words have passed this exercise. They should be repeatedly displayed again and again until the user gives correct answer to them.
4. Answer comparison should be case insensitive: ***BeGin*** and ***begin*** is the same word.
5. for the word/phrase containing any of symbols:
    a. . - dot
    b. , - comma
    c. : - colon
    d. ; - semicolon
    e. / - slash
    f. \ - back slash
    g. (or) - brackets
    h. ? - question mark
    i. ! - exclamation mark

   each of this symbol could be replaced with any other symbol, and the answer also should be treated as correct. For example, for the word ***begin / began / begun*** correct answers are:

    - begin, began, begun
    - begin,began,begun
    - begin (began, begun)
    - begin\began\begun
    - etc.

  But incorrect answers are:
    - begun / began / begin
    - begun \ began \ begin
    - begun (begin / began)
    - etc.

Screen 11

The screen is displayed when the lesson is finished.

1. Display lesson status

1. Percent of correct answers for the lesson is displayed. It is calculated as *correct_answers * 100 / (correct_answers + incorrect_answers)*
2. If correct answers >= 90% then the state “Excellent” is displayed (iphone_11_end_lesson.png). If correct answers < 90% and >= 80%, the state “Good” is displayed instead of “Excellent”. Otherwise no state is displayed.

2. Display words in the lesson

1. List of words passed through the lesson is displayed. Each word contains “error” or “correct” icon. Word has the “error” icon if it has at least 1 error in any of passed exercises. Otherwise word is treated as “correct”.
2. When pressing on the word, sound is displayed.

Screen 12

The screen is displayed when the user selected too small number of words for the lesson. The minimal number of words allowed is 7. Words which have status “learned” and “postponed” could not be used in exercises, only words with status “learn” are counted.

Spaced Repetition Algorithm

As mentioned before, spaced repetition algorithm calculates optimal time intervals for the cards between the lessons. Below is the description of algorithm written in PHP (C-like syntax so it will be clear for C/C++, Java developers)

0) Card object should have the next fields for spaced repetition to take place:

```
private $ease_factor; - depicts how easy is this card for the user
private $prev_ease_factor; - ease factor for the previous lesson
```

```
    private $next_interval; - current repetition interval (in days)
    private $prev_interval; - repetition interval for the previous lesson
    private $lesson_number; - number of passed lessons
    private $yes_count; - number of correct answers
    private $no_count; - number of incorrect answers
    private $succ_yes_count; - number of successive correct answers (or 0)
    private $next_lesson_time; - UNIX time of the next lesson
    private $last_lesson_time; - UNIX time of the last passed lesson
    private $first_lesson_time; - UNIX time of the first lesson for this card
    private $last_change_time; - UNIX time of any last change of the card
```

There are also some important constants for the Card object:

```
    public static $MATURE_THRESHOLD = 21;
    public static $MAX_SCHEDULE_TIME = 36500;

    public static $MAX_EASE_FACTOR = 2.5;
    public static $MIN_EASE_FACTOR = 1.3;
    private static $FACTOR_BOOSTER = 1.3;

    public static $INTERVAL_HARD = 1;
    public static $INTERVAL_MIDIUM_MIN = 3;
    public static $INTERVAL_MIDIUM_MAX = 5;
    public static $INTERVAL_EASY_MIN = 7;
    public static $INTERVAL_EASY_MAX = 9;

    public static $STATUS_LEARN = 1;
    public static $STATUS_LEARNED = 3;
    public static $STATUS_POSTPONE = 4;
```

1) When the lesson is passed, we should have associative array on the output which should look like *errors_by_exercises = array(exercise_number => is_error).* For example for the word ***tree*** we have the array [0, 1, 1, 0]. It means that there were 4 exercises in the lesson and the user made an error in 2nd and 3rd exercise (1 - error, 0 - correct answer)

2) Based on the error array *errors_by_exercises*, we could calculate the *ease* of the card. It could be of 4 types, starting from the easiest:

```
    public static $EASE_EASY = 4;
    public static $EASE_MEDIUM = 3;
    public static $EASE_HARD = 2;
    public static $EASE_NEED_LEARN = 1;
```

The method of getting the *ease* by *errors_by_exercises* array:

```
/**
     * Gets ease of the user's answer according to the stats of
     * correct/incorrect answers by lesson exercises.
     *
     * @param array $errors_by_exercises - stats in format exercise_id => 0|1
     * (1 - there was an error in the exercise, 0 - no errors in the exercise).
     * @return int
     */
```

```
    public static function getEase( array $errors_by_exercises )
    {
        $errors = 0;
        $exerc_count = 0;

        foreach( $errors_by_exercises as $is_error )
        {
            if( $is_error == '1' )
            {
                $errors++;
                $exerc_count++;
            }
            elseif( $is_error == '0' )
            {
                $exerc_count++;
            }
        }

        if( $exerc_count == 0 )
        {
            return self::$EASE_NEED_LEARN;
        }

        $error_level = round( ( $exerc_count - $errors ) * 100 / $exerc_count );

        if( $error_level >= 90 )
        {
            return self::$EASE_EASY;
        }
        elseif( $error_level >= 70 )
        {
            return self::$EASE_MEDIUM;
        }
        elseif( $error_level >= 50 )
        {
            return self::$EASE_HARD;
        }
        else
        {
            return self::$EASE_NEED_LEARN;
        }
    }
```

3) Knowing the ease for the card, we can calculate new repetition interval for the next lesson:

```
public function updateLessonStats( $ease, RsLessonSettings $lessonSettings )
    {
        // get current UNIX time in seconds
        $time = time();

        // flag which means that current lesson is much earlier then scheduled
```

```
$is_too_early = true;

// if current lesson is not the first and earlier then scheduled
if( $this->last_lesson_time != null && $this->next_lesson_time != null &&
    $time < $this->next_lesson_time )
{
    // % of earliness of the current lesson
    $early_rate = ( $this->next_lesson_time - $time ) /
        ( $this->next_lesson_time - $this->last_lesson_time );

    // if earliness value < 30%
    if( $early_rate < 0.3 )
    {
        // not too early
        $is_too_early = false;
    }
}
else
{
    // not too early
    $is_too_early = false;
}

if( $ease == self::$EASE_NEED_LEARN )
{
    $this->succ_yes_count = 0;
    $this->no_count++;
}
elseif( !$is_too_early )
{
    if( $this->succ_yes_count < 255 )
    {
        $this->succ_yes_count++;
    }

    $this->yes_count++;
}

if( $this->isNew() )
{
    $this->first_lesson_time = $time;
    $this->lesson_number = 1;
}
elseif( !$is_too_early )
{
    $this->lesson_number++;
}

$this->last_lesson_time = $time;
$this->last_change_time = $time;
```

```
        $this->updateEaseFactor( $ease );
        $this->prev_interval = $this->next_interval;
        $this->next_interval = $this->nextInterval( $ease );
        $this->next_lesson_time = $time + $this->next_interval * 86400;

        // if card status should be changed
        if( $this->_getProgress( $lessonSettings ) == 100 )
        {
            $this->status_id = RsDictsHelper::$STATUS_LEARNED;
        }
        else
        {
            $this->status_id = RsDictsHelper::$STATUS_LEARN;
        }

        RsDBConnect::getInstance()->run( self::$QUERY_UPDATE_LESSON_STATS,
            array(
                $this->status_id,
                $this->ease_factor,
                $this->prev_ease_factor,
                $this->next_interval,
                $this->prev_interval,
                $this->lesson_number,
                $this->succ_yes_count,
                $this->yes_count,
                $this->no_count,
                $this->next_lesson_time,
                $this->last_lesson_time,
                $this->first_lesson_time,
                $this->last_change_time,
                $this->dict_id,
                $this->user_id,
                $this->phrase_id
            )
        );
    }
```

4) We use some methods in a previous one, which define the “age” of the card. These methods are used quite often across the whole algorithm. Here are they:

```
/**
     * Checks if the card is new meaning that it has not been learned before.
     *
     * @return bool
     */
    public function isNew()
    {
        return $this->lesson_number == 0 || $this->lesson_number == null;
    }

    /**
```

```
     * Checks if the card is under active acquisition.
     *
     * @return bool
     */
    public function isBeingLearned()
    {
        return $this->prev_interval < self::$INTERVAL_EASY_MIN;
    }

    /**
     * Checks if the card is in the young phase of active learn-recall
     *
     * @return bool
     */
    public function isYoung()
    {
        return !$this->isNew() && !$this->isMature();
    }

    /**
     * Checks if the card is in the mature phase of recalling.
     *
     * @return bool
     */
    public function isMature()
    {
        return $this->next_interval > self::$MATURE_THRESHOLD;
    }
```

5) There is also the *updateEaseFactor()* method in p 3). It calculates the new value for the card's *ease_factor* parameter which in turn helps to define the next repetition interval by the *ease* value:

```
/**
     * Updates ease factor to be able to calculate the next lesson interval.
     *
     * @param int $ease - user's last answer estimation
     */
    private function updateEaseFactor( $ease )
    {
        $this->prev_ease_factor = $this->ease_factor;

        // if card is not learned before
        if( $this->isNew() )
        {
            $this->ease_factor = self::$MAX_EASE_FACTOR;
            return;
        }

        // if previous answer(s) was correct and card is not under acquisition
        if( $this->succ_yes_count > 0 && !$this->isBeingLearned() )
        {
            if( $ease == self::$EASE_NEED_LEARN )
            {
                $this->ease_factor -= 0.25;
            }
            elseif( $ease == self::$EASE_HARD )
            {
                $this->ease_factor -= 0.15;
            }
        }

        if( $ease == self::$EASE_EASY )
        {
            $this->ease_factor += 0.10;
        }

        if( $this->ease_factor < self::$MIN_EASE_FACTOR )
        {
            $this->ease_factor = self::$MIN_EASE_FACTOR;
        }
        elseif( $this->ease_factor > self::$MAX_EASE_FACTOR )
        {
            $this->ease_factor = self::$MAX_EASE_FACTOR;
        }
    }
```

6) Knowing ease_factor, we can calculate next repetition interval. It also happens in p 3)

```
/**
     * Gets next lesson interval (in days) for the given card.
     * Should be called after self::updateEaseFactor() to get
     * proper next interval estimation.
     *
     * @param int $ease - user's last answer estimation
     * @return int
     * @see self::updateEaseFactor()
     */
    private function nextInterval( $ease )
    {
        $delay = $this->getCardDelay();
        $interval = $this->next_interval;

        // if lesson is earlier than scheduled
        if( $delay < 0 )
        {
            $interval += $delay;
            $delay = 0;

            if( $interval < self::$INTERVAL_HARD )
            {
                $interval = 0;
            }
        }

        // if first time lesson
        if( $interval == 0 )
        {
            if( $ease == self::$EASE_HARD || $ease == self::$EASE_NEED_LEARN )
            {
                $interval = self::$INTERVAL_HARD;
            }
            elseif( $ease == self::$EASE_MEDIUM )
            {
                $interval = rand( self::$INTERVAL_MIDIUM_MIN,
                    self::$INTERVAL_MIDIUM_MAX );
            }
            elseif( $ease == self::$EASE_EASY )
            {
                $interval = rand( self::$INTERVAL_EASY_MIN,
                    self::$INTERVAL_EASY_MAX );
            }
        }
        else
        {
            if( $ease == self::$EASE_NEED_LEARN )
            {
                $interval = round( $interval * self::$MIN_EASE_FACTOR );
            }
            elseif( $ease == self::$EASE_HARD )
            {
                $interval = round( ( $interval + $delay / 4 ) *
                    $this->ease_factor );
            }
            elseif( $ease == self::$EASE_MEDIUM )
            {
                $interval = round( ( $interval + $delay / 2 ) *
                    $this->ease_factor );
            }
            elseif( $ease == self::$EASE_EASY )
            {
                $interval = round( ( $interval + $delay ) *
                    $this->ease_factor * self::$FACTOR_BOOSTER );
            }

            // make some fuzz to avoid lesson spikes
            $interval = round( $interval * rand( 95, 105 ) / 100 );
        }

        if( $interval > self::$MAX_SCHEDULE_TIME )
        {
            $interval = self::$MAX_SCHEDULE_TIME;
        }
```

```
        return $interval;
    }
```

Method which defines card’s delay (used abowe):

```
    /**
     * Gets card's next lesson delay in days. Negative value in case of
     * beginning the lesson earlier then scheduled.
     *
     * @return int
     */
    private function getCardDelay()
    {
        if( $this->isNew() )
        {
            return 0;
        }

        return round( ( time() - $this->next_lesson_time ) / 86400 );
    }
```

7) There is also the method *_getProgress(),* used in p 3). It gets current learning progress for the current card. If the progress is 100%, then the *status* of the card should become ***“learned”***, otherwise - ***“learn”*** (see **Object Model** section to refer to the *status*):

```
private function _getProgress( RsLessonSettings $lessonSettings )
    {
        if( $this->yes_count == 0 || $this->lesson_number == 0 ||
            $lessonSettings == null )
        {
            return 0;
        }

        $full_prog = $lessonSettings->getAnswersToLearned();
        $cur_prog = round( $this->yes_count * 100 /
            ( $this->yes_count + $this->no_count ) );

        if( $cur_prog > $full_prog )
        {
            $cur_prog = $full_prog;
        }

        if( $this->lesson_number >= $lessonSettings->getLessonsToLearned() )
        {
            return round( $cur_prog * 100 / $full_prog );
        }

        return round( $this->lesson_number * $cur_prog * 100 /
            ( $full_prog * $lessonSettings->getLessonsToLearned() ) );
    }
```

The RsLessonSettings object, used here and in p 3), contains the user's settings (see **Screen 5**). The most important params are: number of lessons which should be passed to make the card's status "learned" for the given user (possible values are 4, 5, 6, …, 10). Second param - percent of total correct answers, which should be reached to make card "learned" (values are 70, 75, 80, 85, 90, 91, 92, 93, 94, 95, 96, 97, 98%). These 2 params in the above code are returned by the methods *getLessonsToLearned()* and *getAnswersToLearned()* consecutive. Params have values 5 and 80% dy default.

8) There are 2 methods to define cards, which are ready for today's lesson:

```
/**
     * Get minimal cutoff time of Today-time learning interval.
     * If some event was earlier than that value then it is considered to be
     * happened not today but in the past.
     *
     * @return int - UNIX time in seconds
     */
    public static function getMinTodayCutoff()
    {
        return time() - 86400;
    }

    /**
     * Get maximal cutoff time of Today-time learning interval.
     * If some event will be later than that value then it is considered to be
     * not today but in the future.
     *
     * @return int - UNIX time in seconds
     */
    public static function getMaxTodayCutoff()
    {
        return time();
    }
```

Cards between these 2 time moments are ready for today's lesson:

```
/**
     * Checks if given time belongs to Today-time learning interval.
     *
     * @param int $time
     * @return bool
     */
    public static function isToday( $time )
    {
        return $time >= self::getMinTodayCutoff() &&
                $time <= self::getMaxTodayCutoff();
    }
```

Overdue cards:

```
    public function isOverdue()
    {
        return $this->next_lesson_time != null &&
            $this->next_lesson_time < self::getMinTodayCutoff();
    }
```

Postponed cards:

```
public function isPostponed()
    {
        return $this->status_id == RsDictsHelper::$STATUS_POSTPONE;
    }
```

Not started cards could be defined with the *isNew()* method, described above.

We have described the whole algorithm, but there are also some additional methods:

a) This method should be used for the cards which statuses are forced to put into the “learned” state (see **Screen 3** -> **Edit cards statuses**)

```
/**
     * Force to set card to the "learned" state. Card gets parameters as if
     * it has already obtained "learned" state though natural way, e.g.
     * via passing exercises.
     *
     * @param RsLessonSettings $lessonSettings
     */
    public function setLearned( RsLessonSettings $lessonSettings )
    {
        // if already learned
        if( $this->_getProgress( $lessonSettings ) == 100 )
        {
            // update just status if needed
            if( $this->status_id != RsDictsHelper::$STATUS_LEARNED )
            {
// update status and last_ctange_time
                RsDBConnect::getInstance()->run( self::$QUERY_UPDATE_STATUS,
                    array(
                        RsDictsHelper::$STATUS_LEARNED,
                        time(),
                        $this->dict_id,
                        $this->user_id,
                        $this->phrase_id
                    )
                );
            }

            return;
        }

// calculate next repetition period as in case the user passed all the exercises with no errors
```

```
    // set "learned" status to the card
    $this->status_id = RsDictsHelper::$STATUS_LEARNED;

    // update ease factor as usual
    $this->updateEaseFactor( self::$EASE_EASY );
    $this->prev_interval = $this->next_interval;

    if( $this->isNew() )
    {
        // set next interval as if card is already mature
        $this->next_interval = self::$MATURE_THRESHOLD;
    }
    else
    {
        // next interval as for very easy answer
        $this->next_interval = $this->nextInterval( self::$EASE_EASY );

        if( $this->next_interval < self::$MATURE_THRESHOLD )
        {
            $this->next_interval = self::$MATURE_THRESHOLD;
        }
    }

    // adjust lesson number value
    if( $this->lesson_number < $lessonSettings->getLessonsToLearned() )
    {
        $this->lesson_number = $lessonSettings->getLessonsToLearned();
    }

    if( $this->succ_yes_count < 255 )
    {
        $this->succ_yes_count++;
    }

// force to set the number of correct answers

    // set "yes" count to satisfy 100% of learning progress for the card
    if( $this->no_count == 0 )
    {
        $this->yes_count = 1;
    }
    else
    {
        $this->yes_count = round(
            $this->no_count * $lessonSettings->getAnswersToLearned() /
                ( 100 - $lessonSettings->getAnswersToLearned() ) );
    }

    $time = time();
```

```
        $this->last_lesson_time = $time;
        $this->last_change_time = $time;
        $this->next_lesson_time = $time + $this->next_interval * 86400;

        if( $this->first_lesson_time == null )
        {
            $this->first_lesson_time = $time;
        }

        RsDBConnect::getInstance()->run( self::$QUERY_UPDATE_LESSON_STATS,
            array(
                $this->status_id,
                $this->ease_factor,
                $this->prev_ease_factor,
                $this->next_interval,
                $this->prev_interval,
                $this->lesson_number,
                $this->succ_yes_count,
                $this->yes_count,
                $this->no_count,
                $this->next_lesson_time,
                $this->last_lesson_time,
                $this->first_lesson_time,
                $this->last_change_time,
                $this->dict_id,
                $this->user_id,
                $this->phrase_id
            )
        );
    }
```

b) Method to release the state of the card as if it is “not started”:

```
/**
     * Resets card's lesson parameters and statistics so card becomes
     * clear as if it wasn't in a learning process at all.
     *
     * @param bool $update_db - update DB or not. True by default.
     */
    public function resetLessonStats( $update_db = true )
    {
        $this->status_id = RsDictsHelper::$STATUS_LEARN;
        $this->ease_factor = self::$MAX_EASE_FACTOR;
        $this->prev_ease_factor = 0;
        $this->next_interval = 0;
        $this->prev_interval = 0;
        $this->lesson_number = 0;
        $this->succ_yes_count = 0;
        $this->yes_count = 0;
        $this->no_count = 0;
        $this->next_lesson_time = null;
        $this->last_lesson_time = null;
        $this->first_lesson_time = null;
        $this->last_change_time = time();

        if( !$update_db )
        {
            return;
        }

        RsDBConnect::getInstance()->run( self::$QUERY_UPDATE_LESSON_STATS,
            array(
                $this->status_id,
                $this->ease_factor,
                $this->prev_ease_factor,
                $this->next_interval,
                $this->prev_interval,
                $this->lesson_number,
                $this->succ_yes_count,
                $this->yes_count,
                $this->no_count,
                $this->next_lesson_time,
                $this->last_lesson_time,
                $this->first_lesson_time,
                $this->last_change_time,
                $this->dict_id,
                $this->user_id,
                $this->phrase_id
            )
        );
    }
```

в) The method helps to output times of previous and next lessons in format “in 3 mon”, “in 2 d”, “3 d overdue” etc:

```
public function getNextLessonTimeFormatted()
    {
        if( $this->next_lesson_time == null )
        {
            return null;
        }

        $interval = null;
        $max_cutoff = self::getMaxTodayCutoff();
        $min_cutoff = self::getMinTodayCutoff();

        if( $this->next_lesson_time > $max_cutoff )
        {
            // positive - in the future
            $interval = $this->next_lesson_time - $max_cutoff;
        }
        elseif( $this->next_lesson_time < $min_cutoff )
        {
            // negative - in the past
            $interval = $this->next_lesson_time - $min_cutoff;
        }
        else
        {
            return gettext( 'today' );
        }

        return $this->formatNextLessonTime( $interval );
    }

public function getLastLessonTimeFormatted()
    {
        if( $this->last_lesson_time == null )
        {
            return null;
        }

        return self::formatTime( $this->last_lesson_time );
    }

private function formatNextLessonTime( $interval )
    {
        $is_in_past = $interval < 0 ? true : false;

        if( $is_in_past )
        {
            // make positive
            $interval = -1 * $interval;
        }

        $months = intval( $interval / 2592000 );
        $days = 0;
        $hours = 0;
        $mins = 0;
        $secs = 0;

        if( $months > 0 )
        {
            $interval = intval( $interval % 2592000 );
        }

        $days = intval( $interval / 86400 );

        if( $days > 0 )
        {
            $interval = intval( $interval % 86400 );
        }

        $hours = intval( $interval / 3600 );
        if( $hours > 0 )
        {
            $interval = intval( $interval % 3600 );
        }

        $mins = intval( $interval / 60 );

        if( $mins > 0 )
        {
            $interval = intval( $interval % 60 );
        }

        $secs = $interval;

        if( $months > 0 )
        {
            return sprintf( $is_in_past ?
                gettext( '%s mon overdue' ) : gettext( 'in %s mon' ), $months );
        }

        if( $days > 0 )
        {
            return sprintf( $is_in_past ?
                gettext( '%s d overdue' ) : gettext( 'in %s d' ), $days );
        }

        if( $hours > 0 )
        {
            return sprintf( $is_in_past ?
                gettext( '%s hr overdue' ) : gettext( 'in %s hr' ), $hours );
        }

        if( $mins == 0 )
        {
            $mins = 1;
        }

        return sprintf( $is_in_past ?
                gettext( '%s min overdue' ) : gettext( 'in %s min' ), $mins );
    }

public static function formatTime( $time )
    {
        $time = time() - $time;

        if( $time < 0 )
        {
            return 0;
        }
```

```
$months = intval( $time / 2592000 );
$days = 0;
$hours = 0;
$mins = 0;
$secs = 0;

if( $months > 0 )
{
    $time = intval( $time % 2592000 );
}

$days = intval( $time / 86400 );

if( $days > 0 )
{
    $time = intval( $time % 86400 );
}

$hours = intval( $time / 3600 );

if( $hours > 0 )
{
    $time = intval( $time % 3600 );
}

$mins = intval( $time / 60 );

if( $mins > 0 )
{
    $time = intval( $time % 60 );
}

$secs = $time;

if( $months > 0 )
{
    return sprintf( gettext( '%s mon ago' ), $months );
}

if( $days > 0 )
{
    return sprintf( gettext( '%s d ago' ), $days );
}

if( $hours > 0 )
{
    return sprintf( gettext( '%s hr ago' ), $hours );
}

if( $mins > 0 )
{
```

```
            return sprintf( gettext( '%s min ago'), $mins );
        }

        return sprintf( gettext( '%s sec ago' ), $secs );
    }
```

Addendum

Available learning and translation languages

List of available languages and their IDs:

6 = Arabic
17 = Chinese
19 = Czech
2 = English
23 = Esperanto
28 = Finnish
29 = French
32 = German
33 = Greek
35 = Hebrew
36 = Hindi
37 = Hungarian
40 = Italian
41 = Japanese
56 = Norwegian
27 = Persian
59 = Polish
61 = Portuguese
1 = Russian
72 = Spanish
75 = Swedish
81 = Thai
84 = Turkish