Behind the Scenes

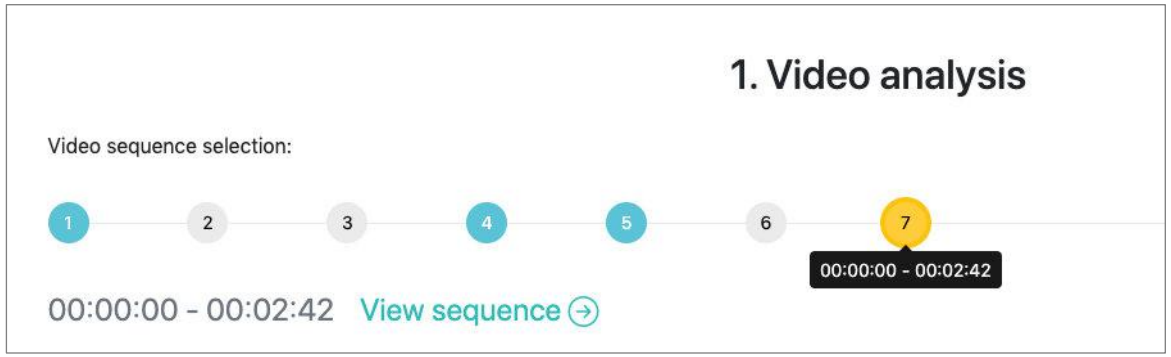
Code & Screenshots

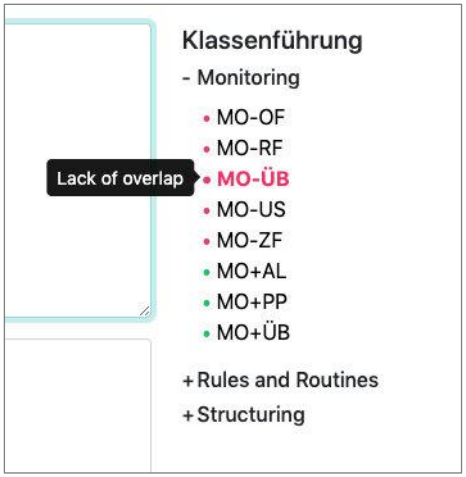
The following pages showcase a selection of distinct processes and application parts.

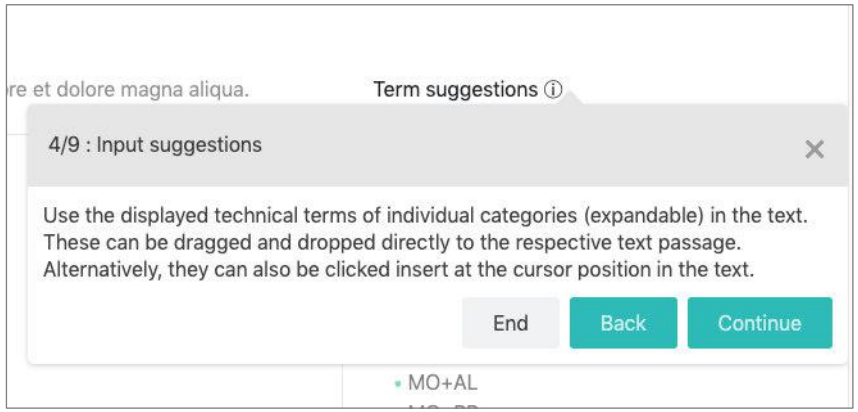
This unveils a few of the underlying processes and features.

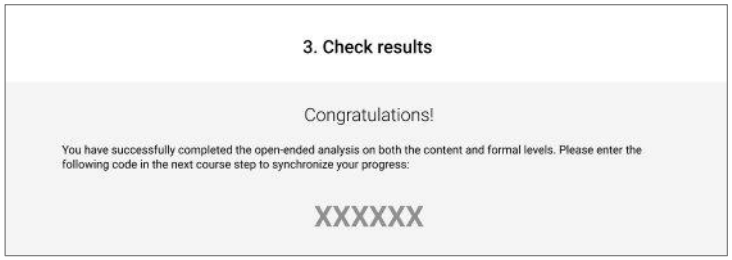
Frontend / UI

App screens – Frontend









Code & Data

App screens – Frontend (Code)

```
/**
 * Main action for application.
 * - Request API data and pass result to view.
 * - Initialize all modules after view is ready.
 *
 * @private
 */
indexAction() {
  const promise = this.pageService.getContent();

  this.authenticationHelper.init();
  this.compatibilityHelper.init();

  promise
    .then((apiData) => this.createPageModel(apiData))
    .then((pageModel) => this.hydrateView(pageModel))
    .then((pageModel) => this.initModules(pageModel))
    .then(() => this.publishReady())
    .catch((error) => {
      throw new Error(error.stack);
    });
}
```

```
/**
 * Get matching words and terms for text.
 * Sort and filter for unique values.
 *
 * @param {String} text
 * @return {Array}
 * @private
 */
getMatchingWordsAndTerms(text) {
  const terms = Object.entries(this.terms);
  const result = [];

  for (const [key, val] of terms) {
    const isMatch = key.toLowerCase().startsWith(text);

    if (isMatch) {
      result.push( ...val);
    }
  }

  return this.getSortedUniqueResult(result);
}
```

```
/**
 * Get if character behind caret position is empty.
 * Check if is empty (no character, except whitespace/line breaks).
 *
 * @example
 *   Examples (Caret position '|' for clarity, not actually in text):
 *   - 'Text|' => true
 *   - 'Te|xt' => false
 * @param {String} text
 * @param {Number} position 0-N
 * @return {Boolean}
 * @private
 */
isWordEndAfterCaret(text, position) {
  const charAfter = text.slice(position, position + 1);

  return charAfter.trim().length === 0;
}
```

```
/**
 * Update existing state with new ones.
 * Merge default with previous and latest state.
 *
 * @param {Object} nextState Arbitrary data
 * @public
 */
update(nextState) {
  const previousState = this.load();

  this.state = {
    ...this.state,
    ...(previousState || {}),
    ...(nextState || {}),
  };

  this.save();
}
```

```
/**
 * Set text on drag start that will be the result on drop.
 *
 * @param {Object} event
 * @private
 */
onDragStart(event) {
  const text = this.sanitizerLib.sanitize(event.target.textContent);

  event.dataTransfer.setData('text/plain', text);
}
```

```
/**
 * Send HTTP request for URL and parameters in given mode via 'fetch'.
 *
 * @param {String} url
 * @param {Object} options
 * @return {Object} Promise HTTP request
 * @private
 */
request(url, options) {
  const urlWithParams = this.getUrlWithParams(url);

  return fetch(urlWithParams, options)
    .then((response) => response.json())
    .catch((error) => {
      throw new Error(error.stack);
    });
}
```

App screens – Frontend (Code)

```
/**
 * Replace form input element value with new text.
 *
 * Steps
 * 1. Get text before caret
 * 2. Match for arrow token (→)
 * 3.1. IF text has arrow token, it's a term: Append the given activity behind (AA→BB)
 * 3.2. Set return value to be empty (as there can nothing come behind a completed term)
 * 4.1. IF text has NO arrow token, it's a word: Overwrite it with a shortcut (Word → AA)
 * 4.2. Set return value to input activity (used to find counterparts of the term as suggestions)
 * 5. Mutate the text field and model state to set new text and caret details
 *
 * @todo Decide - Terms without arrow are recognized as words and should not have empty space after them
 * @param {String} activityShortcut Shortcut name identifier
 * @return {String}
 * @private
 */
getAndReplaceTextData(activityShortcut) {
  const { caretPosition, text } = this.textCaretModelRef.data;
  const termAtPosition = this.termTextSearchLib.getTermAtPosition(text, caretPosition);
  let result = '';

  if (this.hasArrowInText()) {
    this.updateTextFromTerm(activityShortcut);
  } else {
    this.updateTextFromWord(activityShortcut, termAtPosition);
    result = activityShortcut;
  }

  return result;
}
```

```
/**
 * Get terms from shortcuts containing the arrow token.
 *
 * @example
 *   'A'    : No matches
 *   'A→'   : Matches
 *   '→A'   : No matches
 *   'A→B'  : No matches
 *
 * @param {String} term
 * @return {Array} Term identifiers [string]
 * @private
 */
getTermsFromArrowTokens(term) {
  const normalizedTerm = this.getNormalizedTermBeforeSearch(term);
  const tokenMatches = this.arrowTokenMatchLib.getDetails(normalizedTerm);
  let counterpartTerms = [];

  if (tokenMatches.arrowPosition === 'end') {
    const termText = tokenMatches.terms[0] ?? '';

    counterpartTerms = this.listCounterpartSearchLib.getCounterpartTerms(termText);
  }

  return counterpartTerms;
}
```

```
 * we choose a mixed approa
 *
 * 1. Let DomPurify clean the whole text in general.
 * 2. Fully remove the '<' bracket entity and add a whitespace.
 *    - Goal: Not accidentally combine words that should be separate, guessing users might use it here.
 * 3. Convert the '>' bracked back to readable form.
 *
 * @param {String} Text data from storage
 * @return {String}
 * @public
 */
getConvertedSanitizedText(text) {
  const sanitizedText = this.sanitizerLib.sanitize(text);

  return sanitizedText.replace(/&lt;/g, ' ').replace(/&gt;/g, '>');
}
```

App screens – Backend (Code)

```sql
DROP TABLE IF EXISTS `APPNAME_video_details`;
CREATE TABLE `APPNAME_video_details` (
  `id` SERIAL,
  `id_video` BIGINT UNSIGNED NOT NULL,
  `is_timeless` TINYINT NOT NULL,
  `time_from` TIME,
  `time_to` TIME,
  `description` TEXT NOT NULL,
  UNIQUE(id_video, is_timeless, time_from, time_to),
  FOREIGN KEY (`id_video`) REFERENCES APPNAME_videos(`id`),
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```sql
INSERT INTO `APPNAME_activities_terms` (`id`, `term`, `shortcuts`) VALUES
  (NULL, 'Allgegenwärtigkeit', 'MO+AL'),
  (NULL, 'Beschäftigungsradius', 'ST+BR'),
  (NULL, 'Durchgesetzt', 'RR+DS RR-DS'),
  (NULL, 'Durchsetzen', 'RR+DS RR-DS'),
  (NULL, 'Durchsetzung', 'RR+DS RR-DS'),
  (NULL, 'einführen', 'RR+EF RR-EF'),
  (NULL, 'Einführung', 'RR+EF RR-EF'),
  (NULL, 'eingeführt', 'RR+EF RR-EF'),
  (NULL, 'etablieren', 'RR+ET RR-ET'),
  (NULL, 'Etabliertheit', 'RR+ET RR-ET'),
  (NULL, 'geregelt', 'RR+DS RR+EF RR+ET RR-DS RR-EF RR-ET'),
```

```sql
DROP TABLE IF EXISTS `APPNAME_video_details_urls`;
CREATE TABLE `APPNAME_video_details_urls` (
  `id` SERIAL,
  `id_video` BIGINT UNSIGNED NOT NULL,
  `id_video_detail` BIGINT UNSIGNED NOT NULL,
  `id_order` BIGINT UNSIGNED NOT NULL,
  `data_format` ENUM ("MP4", "OGV", "WEBM") NULL,
  `url` VARCHAR(512) NOT NULL UNIQUE,
  UNIQUE(id_video_detail, id_order, data_format),
  FOREIGN KEY (`id_video`) REFERENCES APPNAME_videos(`id`),
  FOREIGN KEY (`id_video_detail`) REFERENCES APPNAME_video_details(`id`),
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```php
/**
 * Unsafe input sanitization.
 */
// phpcs:ignore
interface ISanitization
{
    /**
     * Sanitize single input.
     *
     * @param mixed $input
     * @return mixed
     */
    public function clean($input);

    /**
     * Sanitize many inputs.
     */
    public function cleanAll(array $inputs) : array;
}
```

```php
/**
 * Get sanitized content.
 *
 * @param array $content [[[string] => string]]
 * @return array [[[string] => string]]
 */
private function getSanitizedContent(array $content) : array
{
    foreach ($content as $index => $fields) {
        foreach ($fields as $key => $value) {
            $fields[$key] = $this->csvSanitizer->clean($value);
        }

        $content[$index] = $fields;
    }

    return $content;
}
```

```php
/**
 * Check input and get results.
 */
public function action()
{
    $qualityModel   = $this->qualityModel;
    $postData       = $this->request->getPostData();
    $checkResults   = $qualityModel->getCheckResults($postData);
    $hasPassedTests = $qualityModel->hasPassedTests($checkResults);

    $responseData = $checkResults;

    if ($hasPassedTests) {
        $this->respondSuccess($responseData);
    } else {
        $this->response->send('fail', $responseData);
    }
}
```

App screens – Backend (Code)

```php
/**
 * Sanitize string input:
 * - Strip slashes, e.g. \" ⇒ "
 * - Remove linebreaks that can break CSV
 * - Replace unwanted quotes
 * - Trim
 */
public function clean(string $raw) : string
{
    $unescaped = stripcslashes($raw);
    $unescaped = stripslashes($unescaped);

    $noLinebreaks     = $this->removeLinebreaks($unescaped);
    $consistentQuotes = $this->convertQuotes($noLinebreaks);
    $sanitizedText    = $this->sanitizeCodeInjection($consistentQuotes);

    return trim($sanitizedText);
}
```

```php
 * @example Possible formula injection:
 *   Text before | Text after
 *   =CMD()      | CMD()
 */
private function sanitizeCodeInjection(string $text) : string
{
    $escapeRegex = '/^' . $this->regexNonWordSpecialCharacters . '+/im';

    return preg_replace($escapeRegex, '', $text);
}
```

```php
/**
 * Get texts from database for video category depending on if it's 'interacting' or not.
 */
private function getTextsFromDatabase(int $id) : array
{
    $tablePrefix = $this->tablePrefix;

    $conditional = "(`video_category` IS NULL OR `video_category` =
      (SELECT `{$tablePrefix}videos`.category FROM `{$tablePrefix}videos` WHERE id = {$id}))";

    return $this->queryDatabase('video_texts', [
        'slug',
        'content',
    ], $conditional);
}
```

```php
/**
 * Expect success results for correct test data input.
 */
public function testSuccessResultForCorrectData() : void
{
    $formData = [
        'content_test__interpretation_1'     ⇒ 'Bereits zu Beginn des Videos wir
        'content_test__evaluation_1'         ⇒ 'BEGRUEND_B - da - deut * - dien
        'content_test__alternative_action_1' ⇒ 'BEGRUEND_B - da - deut * - dien
        'content_meta__video_time_id_1'      ⇒ '1',
        'meta_video__id'                     ⇒ '1',
    ];

    $result = self::$instance->getCheckResults($formData);

    $this->assertEquals($result['content']['interpretation']['status'], true);
    $this->assertEquals($result['global']['content']['status'], true);
    $this->assertEquals($result['global']['formal']['status'], true);
    $this->assertEquals($result['formal']['alternativeAction']['status'], true);
    $this->assertEquals($result['formal']['evaluation']['status'], true);
    $this->assertEquals($result['formal']['interpretation']['status'], true);

    $this->assertCount(0, $result['content']['interpretation']['messages']);
    $this->assertCount(0, $result['formal']['alternativeAction']['messages']);
    $this->assertCount(0, $result['formal']['evaluation']['messages']);
    $this->assertCount(0, $result['formal']['interpretation']['messages']);
}
```