```
// src/codec/frameDecoder.ts

import { WnspFrame } from "../protocol/frameTypes";
import {
  decodeWavelengthsToLetters,
  decodeLettersToText,
} from "./textEncoder";
import { computeChecksum, DEFAULT_SYNC_PATTERN }
from "./frameEncoder";

/**
 * Validate a frame using checksum and (optionally) sync
pattern.
 */
export function isValidFrame(
  frame: WnspFrame,
  options?: { expectedSync?: number }
): boolean {
  const expectedSync = options?.expectedSync ??
DEFAULT_SYNC_PATTERN;
  if (frame.sync !== expectedSync) return false;

  const checksum = computeChecksum(
    frame.sync,
    frame.wavelengthNm,
    frame.intensityLevel,
    frame.payloadBit
  );

  return checksum === frame.checksum;
```

```
}

/**
 * Filter out invalid frames from a sequence.
 */
export function filterValidFrames(
  frames: WnspFrame[],
  options?: { expectedSync?: number }
): WnspFrame[] {
  return frames.filter((f) => isValidFrame(f, options));
}

/**
 * Decode frames into a wavelength sequence (nm), ignoring
invalid frames.
 */
export function decodeFramesToWavelengths(
  frames: WnspFrame[],
  options?: { expectedSync?: number }
): number[] {
  const valid = filterValidFrames(frames, options);
  return valid.map((f) => f.wavelengthNm);
}

/**
 * Decode frames directly to letters via wavelength mapping.
 */
export function decodeFramesToLetters(
  frames: WnspFrame[],
  options?: { expectedSync?: number }
```

```
): string[] {
  const wavelengths = decodeFramesToWavelengths(frames,
options);
  return decodeWavelengthsToLetters(wavelengths);
}

/**
 * Decode frames all the way back to text.
 */
public function decodeFramesToText(
  frames: WnspFrame[],
  options?: { expectedSync?: number }
): string {
  const letters = decodeFramesToLetters(frames, options);
  return decodeLettersToText(letters);
}
```