.

PDF .

1.

1.1

```
reporting_system/
├─── frontend/
│    ├─── components/
│    │    ├─── ReportGenerator.js
│    │    ├─── ReportTemplate.js
│    │    ├─── ReportList.js
│    │    └─── ReportViewer.js
│    └─── pages/
│         └─── Reports.js
├─── backend/
│    ├─── routes/
│    │    └─── reports.js
│    ├─── controllers/
│    │    └─── reportController.js
│    ├─── models/
│    │    └─── Report.js
│    ├─── services/
│    │    ├─── pdfGenerator.js
│    │    └─── reportService.js
│    └─── templates/
│         └─── report_template.html
└─── database/
     └─── schema/
          └─── reports.sql
```

1.2

1.
2.
3.

4. PDF
5.
6.

2.

2.1 (ReportGenerator.js)

```
 import React, { useState } from 'react';
import {
  Box,
  Paper,
  Typography,
  TextField,
  Button,
  Grid,
  FormControl,
  InputLabel,
  Select,
  MenuItem,
  Chip,
  CircularProgress,
  Alert,
  Divider
} from '@mui/material';
import { useTranslation } from 'react-i18next';
import { createReport } from '../services/reportService';

const ReportGenerator = ({ analysisResults, projectId, onReportCreated }) => {
  const { t } = useTranslation();
  const [reportData, setReportData] = useState({
    title: '',
    description: '',
    location: '',
    inspectionDate: new Date().toISOString().split('T')[0],
    severity: 'medium',
    recommendations: '',
    tags: []
  });
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');
  const [success, setSuccess] = useState(false);
  const [tag, setTag] = useState('');

  const handleChange = (e) => {
    const { name, value } = e.target;
    setReportData(prev => ({ ...prev, [name]: value }));
  };
```

```
  const handleAddTag = () => {
    if (tag && !reportData.tags.includes(tag)) {
      setReportData(prev => ({ ...prev, tags: [...prev.tags, tag] }));
      setTag('');
    }
  };

  const handleRemoveTag = (tagToRemove) => {
    setReportData(prev => ({
      ...prev,
      tags: prev.tags.filter(t => t !== tagToRemove)
    }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);
    setError('');
    setSuccess(false);

    try {
      //
      const reportPayload = {
        ...reportData,
        projectId,
        analysisResults,
        createdAt: new Date().toISOString()
      };

      //
      const response = await createReport(reportPayload);

      setSuccess(true);
      if (onReportCreated) {
        onReportCreated(response.data);
      }
    } catch (err) {
      setError(t('reports.createError'));
      console.error('Error creating report:', err);
    } finally {
      setLoading(false);
    }
  };

  //
  const determineSeverity = () => {
    if (!analysisResults || !analysisResults.defects || analysisResults.defects.length
=== 0) {
      return 'low';
    }
```

```
    const highConfidenceDefects = analysisResults.defects.filter(d => d.confidence > 0.8);
    const mediumConfidenceDefects = analysisResults.defects.filter(d => d.confidence > 0.5 && d.confidence <= 0.8);

    if (highConfidenceDefects.length > 2 || (highConfidenceDefects.length > 0 && analysisResults.defects.some(d => d.type === 'exposed_rebar'))) {
      return 'high';
    } else if (highConfidenceDefects.length > 0 || mediumConfidenceDefects.length > 2) {
      return 'medium';
    } else {
      return 'low';
    }
  };

  //
  const generateRecommendations = () => {
    if (!analysisResults || !analysisResults.defects || analysisResults.defects.length === 0) {
      return t('reports.recommendations.noDefects');
    }

    const severity = determineSeverity();
    const hasCracks = analysisResults.defects.some(d => d.type === 'crack');
    const hasCorrosion = analysisResults.defects.some(d => d.type === 'corrosion');
    const hasExposedRebar = analysisResults.defects.some(d => d.type === 'exposed_rebar');

    let recommendations = '';

    if (severity === 'high') {
      recommendations += t('reports.recommendations.highSeverity') + '\n\n';
    } else if (severity === 'medium') {
      recommendations += t('reports.recommendations.mediumSeverity') + '\n\n';
    } else {
      recommendations += t('reports.recommendations.lowSeverity') + '\n\n';
    }

    if (hasCracks) {
      recommendations += '- ' + t('reports.recommendations.cracks') + '\n';
    }

    if (hasCorrosion) {
      recommendations += '- ' + t('reports.recommendations.corrosion') + '\n';
    }

    if (hasExposedRebar) {
      recommendations += '- ' + t('reports.recommendations.exposedRebar') + '\n';
    }

    return recommendations;
```

```
  };

  //
  const autoFillData = () => {
    const severity = determineSeverity();
    const recommendations = generateRecommendations();
    const defaultTitle = t('reports.defaultTitle', { date: new
Date().toLocaleDateString() });

    setReportData(prev => ({
      ...prev,
      title: defaultTitle,
      severity,
      recommendations
    }));
  };

  return (
    <Paper sx={{ p: 3, mb: 4 }}>
      <Typography variant="h5" component="h2" gutterBottom>
        {t('reports.generator.title')}
      </Typography>

      {error && (
        <Alert severity="error" sx={{ mb: 3 }}>
          {error}
        </Alert>
      )}

      {success && (
        <Alert severity="success" sx={{ mb: 3 }}>
          {t('reports.createSuccess')}
        </Alert>
      )}

      <Box sx={{ mb: 2, display: 'flex', justifyContent: 'flex-end' }}>
        <Button
          variant="outlined"
          onClick={autoFillData}
          sx={{ mr: 2 }}
        >
          {t('reports.generator.autoFill')}
        </Button>
      </Box>

      <form onSubmit={handleSubmit}>
        <Grid container spacing={3}>
          <Grid item xs={12}>
            <TextField
              fullWidth
              label={t('reports.generator.title')}
              name="title"
```

```
            value={reportData.title}
            onChange={handleChange}
            required
          />
        </Grid>

        <Grid item xs={12} md={6}>
          <TextField
            fullWidth
            label={t('reports.generator.location')}
            name="location"
            value={reportData.location}
            onChange={handleChange}
          />
        </Grid>

        <Grid item xs={12} md={6}>
          <TextField
            fullWidth
            label={t('reports.generator.inspectionDate')}
            name="inspectionDate"
            type="date"
            value={reportData.inspectionDate}
            onChange={handleChange}
            InputLabelProps={{ shrink: true }}
          />
        </Grid>

        <Grid item xs={12}>
          <TextField
            fullWidth
            label={t('reports.generator.description')}
            name="description"
            value={reportData.description}
            onChange={handleChange}
            multiline
            rows={3}
          />
        </Grid>

        <Grid item xs={12}>
          <FormControl fullWidth>
            <InputLabel>{t('reports.generator.severity')}</InputLabel>
            <Select
              name="severity"
              value={reportData.severity}
              onChange={handleChange}
              label={t('reports.generator.severity')}
            >
              <MenuItem value="low">{t('severity.low')}</MenuItem>
              <MenuItem value="medium">{t('severity.medium')}</MenuItem>
              <MenuItem value="high">{t('severity.high')}</MenuItem>
```

```
          </Select>
        </FormControl>
      </Grid>

      <Grid item xs={12}>
        <TextField
          fullWidth
          label={t('reports.generator.recommendations')}
          name="recommendations"
          value={reportData.recommendations}
          onChange={handleChange}
          multiline
          rows={5}
        />
      </Grid>

      <Grid item xs={12}>
        <Box sx={{ mb: 2 }}>
          <Typography variant="subtitle1" gutterBottom>
            {t('reports.generator.tags')}
          </Typography>
          <Box sx={{ display: 'flex', alignItems: 'center' }}>
            <TextField
              label={t('reports.generator.addTag')}
              value={tag}
              onChange={(e) => setTag(e.target.value)}
              sx={{ mr: 1 }}
            />
            <Button
              variant="outlined"
              onClick={handleAddTag}
              disabled={!tag}
            >
              {t('reports.generator.add')}
            </Button>
          </Box>
        </Box>

        <Box sx={{ display: 'flex', flexWrap: 'wrap', gap: 1 }}>
          {reportData.tags.map((tag, index) => (
            <Chip
              key={index}
              label={tag}
              onDelete={() => handleRemoveTag(tag)}
            />
          ))}
        </Box>
      </Grid>

      <Grid item xs={12}>
        <Divider sx={{ my: 2 }} />
        <Box sx={{ display: 'flex', justifyContent: 'flex-end' }}>
```

```
            <Button
              type="submit"
              variant="contained"
              color="primary"
              size="large"
              disabled={loading}
              startIcon={loading ? <CircularProgress size={24} /> : null}
            >
              {loading ? t('reports.generator.creating') : t('reports.generator.create')}
            </Button>
          </Box>
        </Grid>
      </Grid>
    </form>
  </Paper>
 );
};

export default ReportGenerator;
```

2.2 (ReportTemplate.js)

```
 import React from 'react';
import {
  Box,
  Paper,
  Typography,
  Grid,
  Divider,
  Chip,
  Table,
  TableBody,
  TableCell,
  TableContainer,
  TableHead,
  TableRow
} from '@mui/material';
import { useTranslation } from 'react-i18next';

const ReportTemplate = ({ report }) => {
  const { t } = useTranslation();

  if (!report) return null;

  const {
    title,
    description,
    location,
    inspectionDate,
    severity,
```

```
  recommendations,
  tags,
  analysisResults,
  createdAt
} = report;

const formatDate = (dateString) => {
  return new Date(dateString).toLocaleDateString();
};

const getSeverityColor = (severity) => {
  switch (severity) {
    case 'high':
      return '#f44336'; // red
    case 'medium':
      return '#ff9800'; // orange
    case 'low':
      return '#4caf50'; // green
    default:
      return '#2196f3'; // blue
  }
};

return (
  <Paper sx={{ p: 4 }}>
    <Box sx={{ mb: 4 }}>
      <Typography variant="h4" component="h1" gutterBottom>
        {title}
      </Typography>

      <Typography variant="subtitle1" color="textSecondary" gutterBottom>
        {t('reports.template.createdOn')}: {formatDate(createdAt)}
      </Typography>

      <Box sx={{ mt: 2, display: 'flex', flexWrap: 'wrap', gap: 1 }}>
        {tags && tags.map((tag, index) => (
          <Chip key={index} label={tag} size="small" />
        ))}
      </Box>
    </Box>

    <Divider sx={{ mb: 4 }} />

    <Grid container spacing={4}>
      <Grid item xs={12} md={6}>
        <Typography variant="h6" gutterBottom>
          {t('reports.template.details')}
        </Typography>

        <TableContainer component={Paper} variant="outlined" sx={{ mb: 3 }}>
          <Table size="small">
            <TableBody>
```

```
            <TableRow>
              <TableCell component="th" scope="row" sx={{ fontWeight: 'bold' }}>
                {t('reports.template.location')}
              </TableCell>
              <TableCell>{location || t('reports.template.notSpecified')}</TableCell>
            </TableRow>
            <TableRow>
              <TableCell component="th" scope="row" sx={{ fontWeight: 'bold' }}>
                {t('reports.template.inspectionDate')}
              </TableCell>
              <TableCell>{formatDate(inspectionDate)}</TableCell>
            </TableRow>
            <TableRow>
              <TableCell component="th" scope="row" sx={{ fontWeight: 'bold' }}>
                {t('reports.template.severity')}
              </TableCell>
              <TableCell>
                <Chip
                  label={t(`severity.${severity}`)}
                  sx={{
                    bgcolor: getSeverityColor(severity),
                    color: 'white'
                  }}
                  size="small"
                />
              </TableCell>
            </TableRow>
            <TableRow>
              <TableCell component="th" scope="row" sx={{ fontWeight: 'bold' }}>
                {t('reports.template.totalDefects')}
              </TableCell>
              <TableCell>{analysisResults?.total_defects || 0}</TableCell>
            </TableRow>
          </TableBody>
        </Table>
      </TableContainer>

      {description && (
        <Box sx={{ mb: 3 }}>
          <Typography variant="h6" gutterBottom>
            {t('reports.template.description')}
          </Typography>
          <Typography variant="body1" sx={{ whiteSpace: 'pre-line' }}>
            {description}
          </Typography>
        </Box>
      )}
    </Grid>

    <Grid item xs={12} md={6}>
      {analysisResults?.image && (
        <Box sx={{ mb: 3 }}>
```

```
            <Typography variant="h6" gutterBottom>
              {t('reports.template.analyzedImage')}
            </Typography>
            <Box
              component="img"
              src={analysisResults.image}
              alt={t('reports.template.analyzedImage')}
              sx={{
                width: '100%',
                maxHeight: 400,
                objectFit: 'contain',
                border: '1px solid #ddd',
                borderRadius: 1
              }}
            />
          </Box>
        )}
      </Grid>
    </Grid>

    <Divider sx={{ my: 4 }} />

    <Box sx={{ mb: 4 }}>
      <Typography variant="h6" gutterBottom>
        {t('reports.template.defectSummary')}
      </Typography>

      {analysisResults?.defect_summary &&
Object.keys(analysisResults.defect_summary).length > 0 ? (
        <Grid container spacing={2} sx={{ mb: 3 }}>
          {Object.entries(analysisResults.defect_summary).map(([type, count]) => (
            <Grid item xs={12} sm={4} key={type}>
              <Paper
                sx={{
                  p: 2,
                  textAlign: 'center',
                  bgcolor: type === 'crack' ? '#ffebee' :
                        type === 'corrosion' ? '#fff8e1' :
                        type === 'exposed_rebar' ? '#e8f5e9' : '#e3f2fd'
                }}
              >
                <Typography variant="h5" component="div">
                  {count}
                </Typography>
                <Typography variant="body2" color="textSecondary">
                  {t(`defects.${type}`)}
                </Typography>
              </Paper>
            </Grid>
          ))}
        </Grid>
      ) : (
```

```
        <Typography variant="body1">
          {t('reports.template.noDefectsFound')}
        </Typography>
      )}
    </Box>

    {analysisResults?.defects && analysisResults.defects.length > 0 && (
      <Box sx={{ mb: 4 }}>
        <Typography variant="h6" gutterBottom>
          {t('reports.template.detailedDefects')}
        </Typography>

        <TableContainer component={Paper} variant="outlined">
          <Table>
            <TableHead>
              <TableRow>
                <TableCell>{t('reports.template.defectType')}</TableCell>
                <TableCell>{t('reports.template.confidence')}</TableCell>
                <TableCell>{t('reports.template.dimensions')}</TableCell>
                <TableCell>{t('reports.template.severity')}</TableCell>
              </TableRow>
            </TableHead>
            <TableBody>
              {analysisResults.defects.map((defect, index) => {
                const defectSeverity = defect.confidence > 0.8 ? 'high' :
                                       defect.confidence > 0.5 ? 'medium' : 'low';

                return (
                  <TableRow key={index}>
                    <TableCell>{t(`defects.${defect.type}`)}</TableCell>
                    <TableCell>{(defect.confidence * 100).toFixed(0)}%</TableCell>
                    <TableCell>
                      {defect.dimensions ? (
                        `${Math.round(defect.dimensions.width)} × ${
Math.round(defect.dimensions.height)} px`
                      ) : (
                        t('reports.template.notAvailable')
                      )}
                    </TableCell>
                    <TableCell>
                      <Chip
                        label={t(`severity.${defectSeverity}`)}
                        sx={{
                          bgcolor: getSeverityColor(defectSeverity),
                          color: 'white'
                        }}
                        size="small"
                      />
                    </TableCell>
                  </TableRow>
                );
              })}
```

```
            </TableBody>
          </Table>
        </TableContainer>
      </Box>
    )}

    <Divider sx={{ my: 4 }} />

    <Box sx={{ mb: 4 }}>
      <Typography variant="h6" gutterBottom>
        {t('reports.template.recommendations')}
      </Typography>

      <Typography variant="body1" sx={{ whiteSpace: 'pre-line' }}>
        {recommendations || t('reports.template.noRecommendations')}
      </Typography>
    </Box>

    <Box sx={{ mt: 6, pt: 2, borderTop: '1px solid #eee' }}>
      <Typography variant="caption" color="textSecondary">
        {t('reports.template.footer', { date: formatDate(createdAt) })}
      </Typography>
    </Box>
  </Paper>
 );
};

export default ReportTemplate;
```

2.3 (ReportList.js)

```
import React, { useState, useEffect } from 'react';
import {
 Box,
 Paper,
 Typography,
 List,
 ListItem,
 ListItemText,
 ListItemSecondaryAction,
 IconButton,
 Chip,
 Divider,
 TextField,
 InputAdornment,
 CircularProgress,
 Alert,
 Menu,
 MenuItem,
 Button
```

```
} from '@mui/material';
import {
  Search as SearchIcon,
  MoreVert as MoreVertIcon,
  Visibility as VisibilityIcon,
  GetApp as GetAppIcon,
  Share as ShareIcon,
  Delete as DeleteIcon,
  FilterList as FilterListIcon
} from '@mui/icons-material';
import { useTranslation } from 'react-i18next';
import { getReports, deleteReport } from '../services/reportService';

const ReportList = ({ onViewReport, onRefresh }) => {
  const { t } = useTranslation();
  const [reports, setReports] = useState([]);
  const [filteredReports, setFilteredReports] = useState([]);
  const [searchTerm, setSearchTerm] = useState('');
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');
  const [menuAnchorEl, setMenuAnchorEl] = useState(null);
  const [selectedReport, setSelectedReport] = useState(null);

  useEffect(() => {
    fetchReports();
  }, [onRefresh]);

  useEffect(() => {
    if (reports.length > 0) {
      filterReports();
    }
  }, [searchTerm, reports]);

  const fetchReports = async () => {
    setLoading(true);
    setError('');

    try {
      const response = await getReports();
      setReports(response.data);
      setFilteredReports(response.data);
    } catch (err) {
      setError(t('reports.list.fetchError'));
      console.error('Error fetching reports:', err);
    } finally {
      setLoading(false);
    }
  };

  const filterReports = () => {
    if (!searchTerm) {
      setFilteredReports(reports);
```

```
      return;
    }

    const filtered = reports.filter(report =>
      report.title.toLowerCase().includes(searchTerm.toLowerCase()) ||
      (report.description &&
report.description.toLowerCase().includes(searchTerm.toLowerCase())) ||
      (report.location &&
report.location.toLowerCase().includes(searchTerm.toLowerCase())) ||
      (report.tags && report.tags.some(tag =>
tag.toLowerCase().includes(searchTerm.toLowerCase())))
    );

    setFilteredReports(filtered);
  };

  const handleSearchChange = (e) => {
    setSearchTerm(e.target.value);
  };

  const handleMenuOpen = (event, report) => {
    setMenuAnchorEl(event.currentTarget);
    setSelectedReport(report);
  };

  const handleMenuClose = () => {
    setMenuAnchorEl(null);
    setSelectedReport(null);
  };

  const handleViewReport = () => {
    if (selectedReport && onViewReport) {
      onViewReport(selectedReport);
    }
    handleMenuClose();
  };

  const handleDeleteReport = async () => {
    if (!selectedReport) return;

    try {
      await deleteReport(selectedReport.id);
      setReports(reports.filter(r => r.id !== selectedReport.id));
      handleMenuClose();
    } catch (err) {
      setError(t('reports.list.deleteError'));
      console.error('Error deleting report:', err);
    }
  };

  const getSeverityColor = (severity) => {
    switch (severity) {
```

```
      case 'high':
        return '#f44336'; // red
      case 'medium':
        return '#ff9800'; // orange
      case 'low':
        return '#4caf50'; // green
      default:
        return '#2196f3'; // blue
    }
  };

  const formatDate = (dateString) => {
    return new Date(dateString).toLocaleDateString();
  };

  return (
    <Paper sx={{ p: 3 }}>
      <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center',
mb: 3 }}>
        <Typography variant="h6" component="h2">
          {t('reports.list.title')}
        </Typography>

        <Box sx={{ display: 'flex', alignItems: 'center' }}>
          <TextField
            placeholder={t('reports.list.search')}
            value={searchTerm}
            onChange={handleSearchChange}
            size="small"
            sx={{ mr: 2 }}
            InputProps={{
              startAdornment: (
                <InputAdornment position="start">
                  <SearchIcon />
                </InputAdornment>
              ),
            }}
          />

          <Button
            variant="outlined"
            startIcon={<FilterListIcon />}
            size="small"
          >
            {t('reports.list.filter')}
          </Button>
        </Box>
      </Box>

      {error && (
        <Alert severity="error" sx={{ mb: 3 }}>
          {error}
```

```
      </Alert>
    )}

    {loading ? (
      <Box sx={{ display: 'flex', justifyContent: 'center', p: 4 }}>
        <CircularProgress />
      </Box>
    ) : filteredReports.length === 0 ? (
      <Box sx={{ textAlign: 'center', p: 4 }}>
        <Typography variant="body1" color="textSecondary">
          {searchTerm ? t('reports.list.noSearchResults') : t('reports.list.noReports')}
        </Typography>
      </Box>
    ) : (
      <List>
        {filteredReports.map((report, index) => (
          <React.Fragment key={report.id || index}>
            <ListItem alignItems="flex-start">
              <ListItemText
                primary={
                  <Box sx={{ display: 'flex', alignItems: 'center' }}>
                    <Typography variant="subtitle1" component="span">
                      {report.title}
                    </Typography>
                    <Chip
                      label={t(`severity.${report.severity}`)}
                      size="small"
                      sx={{
                        ml: 2,
                        bgcolor: getSeverityColor(report.severity),
                        color: 'white'
                      }}
                    />
                  </Box>
                }
                secondary={
                  <Box sx={{ mt: 1 }}>
                    <Typography variant="body2" color="textSecondary"
component="span">
                      {report.location && `${t('reports.list.location')}: ${report.location} | `}
                      {t('reports.list.date')}: {formatDate(report.createdAt)}
                    </Typography>

                    {report.description && (
                      <Typography
                        variant="body2"
                        color="textSecondary"
                        sx={{
                          mt: 1,
                          display: '-webkit-box',
                          WebkitLineClamp: 2,
                          WebkitBoxOrient: 'vertical',
```

```
                    overflow: 'hidden',
                    textOverflow: 'ellipsis'
                  }}
                >
                  {report.description}
                </Typography>
              )}

              {report.tags && report.tags.length > 0 && (
                <Box sx={{ mt: 1, display: 'flex', flexWrap: 'wrap', gap: 0.5 }}>
                  {report.tags.map((tag, tagIndex) => (
                    <Chip
                      key={tagIndex}
                      label={tag}
                      size="small"
                      variant="outlined"
                    />
                  ))}
                </Box>
              )}
            </Box>
          }
        />
        <ListItemSecondaryAction>
          <IconButton edge="end" onClick={(e) => handleMenuOpen(e, report)}>
            <MoreVertIcon />
          </IconButton>
        </ListItemSecondaryAction>
      </ListItem>
      {index < filteredReports.length - 1 && <Divider />}
    </React.Fragment>
  ))}
</List>
)}

<Menu
  anchorEl={menuAnchorEl}
  open={Boolean(menuAnchorEl)}
  onClose={handleMenuClose}
>
  <MenuItem onClick={handleViewReport}>
    <VisibilityIcon fontSize="small" sx={{ mr: 1 }} />
    {t('reports.list.view')}
  </MenuItem>
  <MenuItem>
    <GetAppIcon fontSize="small" sx={{ mr: 1 }} />
    {t('reports.list.download')}
  </MenuItem>
  <MenuItem>
    <ShareIcon fontSize="small" sx={{ mr: 1 }} />
    {t('reports.list.share')}
  </MenuItem>
```

```
        <Divider />
        <MenuItem onClick={handleDeleteReport} sx={{ color: 'error.main' }}>
          <DeleteIcon fontSize="small" sx={{ mr: 1 }} />
          {t('reports.list.delete')}
        </MenuItem>
      </Menu>
    </Paper>
  );
};

export default ReportList;
```

2.4 (ReportViewer.js)

```
 import React, { useState } from 'react';
import {
  Box,
  Dialog,
  DialogContent,
  DialogActions,
  Button,
  IconButton,
  Tooltip,
  CircularProgress
} from '@mui/material';
import {
  Close as CloseIcon,
  GetApp as GetAppIcon,
  Share as ShareIcon,
  Print as PrintIcon
} from '@mui/icons-material';
import { useTranslation } from 'react-i18next';
import ReportTemplate from './ReportTemplate';
import { downloadReport } from '../services/reportService';

const ReportViewer = ({ report, open, onClose }) => {
  const { t } = useTranslation();
  const [downloading, setDownloading] = useState(false);

  const handleDownload = async () => {
    if (!report) return;

    setDownloading(true);
    try {
      await downloadReport(report.id);
      setDownloading(false);
    } catch (err) {
      console.error('Error downloading report:', err);
      setDownloading(false);
    }
```

```
  };

  const handlePrint = () => {
    window.print();
  };

  return (
    <Dialog
      open={open}
      onClose={onClose}
      maxWidth="lg"
      fullWidth
      scroll="paper"
      aria-labelledby="report-viewer-dialog"
    >
      <Box sx={{ display: 'flex', justifyContent: 'flex-end', p: 1 }}>
        <Tooltip title={t('reports.viewer.print')}>
          <IconButton onClick={handlePrint}>
            <PrintIcon />
          </IconButton>
        </Tooltip>

        <Tooltip title={t('reports.viewer.share')}>
          <IconButton>
            <ShareIcon />
          </IconButton>
        </Tooltip>

        <Tooltip title={t('reports.viewer.download')}>
          <IconButton onClick={handleDownload} disabled={downloading}>
            {downloading ? <CircularProgress size={24} /> : <GetAppIcon />}
          </IconButton>
        </Tooltip>

        <Tooltip title={t('reports.viewer.close')}>
          <IconButton onClick={onClose} edge="end">
            <CloseIcon />
          </IconButton>
        </Tooltip>
      </Box>

      <DialogContent dividers>
        <Box id="printable-report">
          <ReportTemplate report={report} />
        </Box>
      </DialogContent>

      <DialogActions>
        <Button onClick={onClose}>{t('reports.viewer.close')}</Button>
        <Button
          variant="contained"
          startIcon={downloading ? <CircularProgress size={24} /> : <GetAppIcon />}
```

```
          onClick={handleDownload}
          disabled={downloading}
        >
          {downloading ? t('reports.viewer.downloading') : t('reports.viewer.download')}
        </Button>
      </DialogActions>
    </Dialog>
  );
};

export default ReportViewer;
```

2.5 (Reports.js)

```
 import React, { useState, useEffect } from 'react';
import {
  Container,
  Typography,
  Box,
  Tabs,
  Tab,
  Button,
  Paper
} from '@mui/material';
import { Add as AddIcon } from '@mui/icons-material';
import { useTranslation } from 'react-i18next';
import ReportList from '../components/ReportList';
import ReportViewer from '../components/ReportViewer';
import ReportGenerator from '../components/ReportGenerator';

const Reports = () => {
  const { t } = useTranslation();
  const [tabValue, setTabValue] = useState(0);
  const [selectedReport, setSelectedReport] = useState(null);
  const [viewerOpen, setViewerOpen] = useState(false);
  const [showGenerator, setShowGenerator] = useState(false);
  const [refreshTrigger, setRefreshTrigger] = useState(0);

  const handleTabChange = (event, newValue) => {
    setTabValue(newValue);
  };

  const handleViewReport = (report) => {
    setSelectedReport(report);
    setViewerOpen(true);
  };

  const handleCloseViewer = () => {
    setViewerOpen(false);
  };
```

```
  const handleCreateReport = () => {
    setShowGenerator(true);
  };

  const handleReportCreated = () => {
    setShowGenerator(false);
    setRefreshTrigger(prev => prev + 1);
  };

  return (
    <Container maxWidth="lg" sx={{ mt: 4, mb: 4 }}>
      <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center',
mb: 3 }}>
        <Typography variant="h4" component="h1">
          {t('reports.title')}
        </Typography>

        <Button
          variant="contained"
          startIcon={<AddIcon />}
          onClick={handleCreateReport}
        >
          {t('reports.createNew')}
        </Button>
      </Box>

      <Paper sx={{ mb: 3 }}>
        <Tabs
          value={tabValue}
          onChange={handleTabChange}
          indicatorColor="primary"
          textColor="primary"
          variant="fullWidth"
        >
          <Tab label={t('reports.tabs.all')} />
          <Tab label={t('reports.tabs.recent')} />
          <Tab label={t('reports.tabs.shared')} />
        </Tabs>
      </Paper>

      {showGenerator ? (
        <ReportGenerator
          analysisResults={null} //
          projectId={null} //
          onReportCreated={handleReportCreated}
        />
      ) : (
        <ReportList
          onViewReport={handleViewReport}
          onRefresh={refreshTrigger}
        />
```

```
      )}

      <ReportViewer
        report={selectedReport}
        open={viewerOpen}
        onClose={handleCloseViewer}
      />
    </Container>
  );
};

export default Reports;
```

3.

3.1 (Report.js)

```
 // backend/models/Report.js
const mongoose = require('mongoose');

const ReportSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
    trim: true
  },
  description: {
    type: String,
    trim: true
  },
  location: {
    type: String,
    trim: true
  },
  inspectionDate: {
    type: Date,
    default: Date.now
  },
  severity: {
    type: String,
    enum: ['low', 'medium', 'high'],
    default: 'medium'
  },
  recommendations: {
    type: String,
    trim: true
  },
  tags: {
```

```
    type: [String],
    default: []
  },
  analysisResults: {
    type: Object,
    default: null
  },
  projectId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Project',
    default: null
  },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  pdfUrl: {
    type: String,
    default: null
  },
  createdAt: {
    type: Date,
    default: Date.now
  },
  updatedAt: {
    type: Date,
    default: Date.now
  }
});

//              updatedAt
ReportSchema.pre('save', function(next) {
  this.updatedAt = Date.now();
  next();
});

module.exports = mongoose.model('Report', ReportSchema);
```

3.2 PDF (pdfGenerator.js)

```
 // backend/services/pdfGenerator.js
const fs = require('fs');
const path = require('path');
const puppeteer = require('puppeteer');
const handlebars = require('handlebars');
const { v4: uuidv4 } = require('uuid');

//                  Handlebars
handlebars.registerHelper('formatDate', function(date) {
```

```
  return new Date(date).toLocaleDateString();
});

handlebars.registerHelper('json', function(context) {
  return JSON.stringify(context);
});

handlebars.registerHelper('ifEquals', function(arg1, arg2, options) {
  return (arg1 === arg2) ? options.fn(this) : options.inverse(this);
});

/**
 *              PDF
 * @param {Object} report -
 * @returns {Promise<string>} -              PDF
 */
async function generatePDF(report) {
  try {
    //               HTML
    const templatePath = path.join(__dirname, '../templates/report_template.html');
    const templateHtml = fs.readFileSync(templatePath, 'utf8');

    //
    const template = handlebars.compile(templateHtml);
    const html = template({
      report,
      title: report.title,
      date: new Date().toLocaleDateString(),
      baseUrl: process.env.BASE_URL || 'http://localhost:3000'
    });

    //
    const tempDir = path.join(__dirname, '../temp');
    if (!fs.existsSync(tempDir)) {
      fs.mkdirSync(tempDir, { recursive: true });
    }

    //
    const reportsDir = path.join(__dirname, '../public/reports');
    if (!fs.existsSync(reportsDir)) {
      fs.mkdirSync(reportsDir, { recursive: true });
    }

    //
    const filename = `report_${report._id || uuidv4()}.pdf`;
    const outputPath = path.join(reportsDir, filename);

    //              HTML
    const tempHtmlPath = path.join(tempDir, `${uuidv4()}.html`);
    fs.writeFileSync(tempHtmlPath, html);

    //              PDF              Puppeteer
```

```
    const browser = await puppeteer.launch({
      headless: true,
      args: ['--no-sandbox', '--disable-setuid-sandbox']
    });

    const page = await browser.newPage();

    //                          A4
    await page.setViewport({ width: 1240, height: 1754 });
    await page.emulateMediaType('screen');

    //              HTML
    await page.goto(`file://${tempHtmlPath}`, { waitUntil: 'networkidle0' });

    //              PDF
    await page.pdf({
      path: outputPath,
      format: 'A4',
      printBackground: true,
      margin: {
        top: '20mm',
        right: '20mm',
        bottom: '20mm',
        left: '20mm'
      }
    });

    await browser.close();

    //           HTML
    fs.unlinkSync(tempHtmlPath);

    //                   PDF
    return `/reports/${filename}`;
  } catch (error) {
    console.error('Error generating PDF:', error);
    throw new Error('Failed to generate PDF report');
  }
}

module.exports = {
  generatePDF
};
```

3.3 (reportService.js)

```
// backend/services/reportService.js
const Report = require('../models/Report');
const { generatePDF } = require('./pdfGenerator');
const fs = require('fs');
```

```
const path = require('path');

/**
 *
 * @param {Object} reportData -
 * @param {Object} user -
 * @returns {Promise<Object>} -
 */
async function createReport(reportData, user) {
  try {
    //
    const report = new Report({
      ...reportData,
      userId: user._id
    });

    //
    await report.save();

    //              PDF
    const pdfUrl = await generatePDF(report);

    //                                PDF
    report.pdfUrl = pdfUrl;
    await report.save();

    return report;
  } catch (error) {
    console.error('Error creating report:', error);
    throw new Error('Failed to create report');
  }
}

/**
 *
 * @param {string} reportId -
 * @returns {Promise<Object>} -
 */
async function getReportById(reportId) {
  try {
    const report = await Report.findById(reportId);
    if (!report) {
      throw new Error('Report not found');
    }
    return report;
  } catch (error) {
    console.error('Error getting report:', error);
    throw error;
  }
}

/**
```

```
 *
 * @param {Object} user -
 * @param {Object} filters -
 * @returns {Promise<Array>} -
 */
async function getReportsByUser(user, filters = {}) {
  try {
    const query = { userId: user._id };

    //
    if (filters.severity) {
      query.severity = filters.severity;
    }

    if (filters.projectId) {
      query.projectId = filters.projectId;
    }

    if (filters.startDate && filters.endDate) {
      query.createdAt = {
        $gte: new Date(filters.startDate),
        $lte: new Date(filters.endDate)
      };
    }

    //
    const reports = await Report.find(query)
      .sort({ createdAt: -1 })
      .limit(filters.limit || 100);

    return reports;
  } catch (error) {
    console.error('Error getting reports:', error);
    throw new Error('Failed to get reports');
  }
}

/**
 *
 * @param {string} reportId -
 * @param {Object} updateData -
 * @param {Object} user -
 * @returns {Promise<Object>} -
 */
async function updateReport(reportId, updateData, user) {
  try {
    //
    const report = await Report.findOne({ _id: reportId, userId: user._id });
    if (!report) {
      throw new Error('Report not found or unauthorized');
    }
```

```
    //
    Object.keys(updateData).forEach(key => {
      if (key !== '_id' && key !== 'userId' && key !== 'createdAt') {
        report[key] = updateData[key];
      }
    });

    //
    await report.save();

    //                    PDF
    const pdfUrl = await generatePDF(report);
    report.pdfUrl = pdfUrl;
    await report.save();

    return report;
  } catch (error) {
    console.error('Error updating report:', error);
    throw error;
  }
}

/**
 *
 * @param {string} reportId -
 * @param {Object} user -
 * @returns {Promise<boolean>} -
 */
async function deleteReport(reportId, user) {
  try {
    //
    const report = await Report.findOne({ _id: reportId, userId: user._id });
    if (!report) {
      throw new Error('Report not found or unauthorized');
    }

    //          PDF
    if (report.pdfUrl) {
      const pdfPath = path.join(__dirname, '../public', report.pdfUrl);
      if (fs.existsSync(pdfPath)) {
        fs.unlinkSync(pdfPath);
      }
    }

    //
    await Report.deleteOne({ _id: reportId });

    return true;
  } catch (error) {
    console.error('Error deleting report:', error);
    throw error;
  }
```

```
}

/**
 *
 * @param {string} reportId -
 * @param {Array} userIds -
 * @param {Object} user -
 * @returns {Promise<Object>} -
 */
async function shareReport(reportId, userIds, user) {
  //
  throw new Error('Not implemented yet');
}

module.exports = {
  createReport,
  getReportById,
  getReportsByUser,
  updateReport,
  deleteReport,
  shareReport
};
```

3.4 (reportController.js)

```
 // backend/controllers/reportController.js
const reportService = require('../services/reportService');
const path = require('path');
const fs = require('fs');

/**
 *
 * @param {Object} req -       HTTP
 * @param {Object} res -            HTTP
 */
async function createReport(req, res) {
  try {
    const reportData = req.body;
    const user = req.user; //     middleware

    const report = await reportService.createReport(reportData, user);

    res.status(201).json({
      success: true,
      data: report
    });
  } catch (error) {
    console.error('Error in createReport controller:', error);
    res.status(500).json({
      success: false,
```

```
      message: error.message || 'Failed to create report'
    });
  }
}

/**
 *
 * @param {Object} req -       HTTP
 * @param {Object} res -            HTTP
 */
async function getReport(req, res) {
  try {
    const { id } = req.params;
    const user = req.user;

    const report = await reportService.getReportById(id);

    //
    if (report.userId.toString() !== user._id.toString()) {
      return res.status(403).json({
        success: false,
        message: 'Unauthorized access to report'
      });
    }

    res.status(200).json({
      success: true,
      data: report
    });
  } catch (error) {
    console.error('Error in getReport controller:', error);
    res.status(error.message === 'Report not found' ? 404 : 500).json({
      success: false,
      message: error.message || 'Failed to get report'
    });
  }
}

/**
 *
 * @param {Object} req -       HTTP
 * @param {Object} res -            HTTP
 */
async function getReports(req, res) {
  try {
    const user = req.user;
    const filters = req.query;

    const reports = await reportService.getReportsByUser(user, filters);

    res.status(200).json({
      success: true,
```

```
      count: reports.length,
      data: reports
    });
  } catch (error) {
    console.error('Error in getReports controller:', error);
    res.status(500).json({
      success: false,
      message: error.message || 'Failed to get reports'
    });
  }
}

/**
 *
 * @param {Object} req -       HTTP
 * @param {Object} res -            HTTP
 */
async function updateReport(req, res) {
  try {
    const { id } = req.params;
    const updateData = req.body;
    const user = req.user;

    const report = await reportService.updateReport(id, updateData, user);

    res.status(200).json({
      success: true,
      data: report
    });
  } catch (error) {
    console.error('Error in updateReport controller:', error);
    res.status(error.message.includes('not found') ? 404 : 500).json({
      success: false,
      message: error.message || 'Failed to update report'
    });
  }
}

/**
 *
 * @param {Object} req -       HTTP
 * @param {Object} res -            HTTP
 */
async function deleteReport(req, res) {
  try {
    const { id } = req.params;
    const user = req.user;

    await reportService.deleteReport(id, user);

    res.status(200).json({
      success: true,
```

```
      message: 'Report deleted successfully'
    });
  } catch (error) {
    console.error('Error in deleteReport controller:', error);
    res.status(error.message.includes('not found') ? 404 : 500).json({
      success: false,
      message: error.message || 'Failed to delete report'
    });
  }
}

/**
 *                        PDF
 * @param {Object} req -       HTTP
 * @param {Object} res -             HTTP
 */
async function downloadReport(req, res) {
  try {
    const { id } = req.params;
    const user = req.user;

    const report = await reportService.getReportById(id);

    //
    if (report.userId.toString() !== user._id.toString()) {
      return res.status(403).json({
        success: false,
        message: 'Unauthorized access to report'
      });
    }

    //                          PDF
    if (!report.pdfUrl) {
      return res.status(404).json({
        success: false,
        message: 'PDF file not found'
      });
    }

    //              PDF
    const pdfPath = path.join(__dirname, '../public', report.pdfUrl);

    if (!fs.existsSync(pdfPath)) {
      return res.status(404).json({
        success: false,
        message: 'PDF file not found on server'
      });
    }

    res.download(pdfPath, `${report.title.replace(/\s+/g, '_')}.pdf`);
  } catch (error) {
    console.error('Error in downloadReport controller:', error);
```

```
    res.status(error.message === 'Report not found' ? 404 : 500).json({
      success: false,
      message: error.message || 'Failed to download report'
    });
  }
}

module.exports = {
  createReport,
  getReport,
  getReports,
  updateReport,
  deleteReport,
  downloadReport
};
```

3.5 (reports.js)

```
// backend/routes/reports.js
const express = require('express');
const router = express.Router();
const reportController = require('../controllers/reportController');
const { authenticate } = require('../middleware/auth');

//  middleware
router.use(authenticate);

//
router.post('/', reportController.createReport);

//
router.get('/', reportController.getReports);

//
router.get('/:id', reportController.getReport);

//
router.put('/:id', reportController.updateReport);

//
router.delete('/:id', reportController.deleteReport);

//  PDF
router.get('/:id/download', reportController.downloadReport);

module.exports = router;
```

3.6 HTML (report_template.html)

```
<!-- backend/templates/report_template.html -->
<!DOCTYPE html>
<html lang="ar" dir="rtl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{{report.title}}</title>
  <style>
    @import url('https://fonts.googleapis.com/css2?
family=Tajawal:wght@400;500;700&display=swap');

    body {
      font-family: 'Tajawal', Arial, sans-serif;
      margin: 0;
      padding: 0;
      color: #333;
      background-color: #fff;
    }

    .container {
      max-width: 800px;
      margin: 0 auto;
      padding: 20px;
    }

    .header {
      text-align: center;
      margin-bottom: 30px;
      border-bottom: 2px solid #1976d2;
      padding-bottom: 20px;
    }

    .logo {
      max-width: 150px;
      margin-bottom: 10px;
    }

    h1 {
      color: #1976d2;
      margin: 0;
      font-size: 24px;
    }

    .date {
      color: #666;
      font-size: 14px;
      margin-top: 5px;
    }
```

```css
.section {
  margin-bottom: 30px;
}

.section-title {
  color: #1976d2;
  border-bottom: 1px solid #eee;
  padding-bottom: 5px;
  margin-bottom: 15px;
  font-size: 18px;
}

.info-table {
  width: 100%;
  border-collapse: collapse;
  margin-bottom: 20px;
}

.info-table th, .info-table td {
  padding: 8px;
  border: 1px solid #ddd;
  text-align: right;
}

.info-table th {
  background-color: #f5f5f5;
  font-weight: 500;
  width: 30%;
}

.severity {
  display: inline-block;
  padding: 3px 8px;
  border-radius: 4px;
  font-size: 12px;
  color: white;
}

.severity-high {
  background-color: #f44336;
}

.severity-medium {
  background-color: #ff9800;
}

.severity-low {
  background-color: #4caf50;
}

.defect-summary {
  display: flex;
```

```
  justify-content: space-between;
  margin-bottom: 20px;
}

.defect-card {
  flex: 1;
  padding: 15px;
  border-radius: 4px;
  text-align: center;
  margin: 0 5px;
}

.defect-card.crack {
  background-color: #ffebee;
}

.defect-card.corrosion {
  background-color: #fff8e1;
}

.defect-card.exposed_rebar {
  background-color: #e8f5e9;
}

.defect-count {
  font-size: 24px;
  font-weight: bold;
  margin: 5px 0;
}

.defect-type {
  font-size: 14px;
  color: #666;
}

.defects-table {
  width: 100%;
  border-collapse: collapse;
}

.defects-table th, .defects-table td {
  padding: 8px;
  border: 1px solid #ddd;
  text-align: right;
}

.defects-table th {
  background-color: #f5f5f5;
  font-weight: 500;
}

.image-container {
```

```
      text-align: center;
      margin: 20px 0;
    }

    .analyzed-image {
      max-width: 100%;
      max-height: 400px;
      border: 1px solid #ddd;
      border-radius: 4px;
    }

    .recommendations {
      background-color: #f5f5f5;
      padding: 15px;
      border-radius: 4px;
      white-space: pre-line;
    }

    .footer {
      margin-top: 50px;
      padding-top: 20px;
      border-top: 1px solid #eee;
      text-align: center;
      font-size: 12px;
      color: #666;
    }

    .tags {
      margin-top: 10px;
    }

    .tag {
      display: inline-block;
      background-color: #e0e0e0;
      padding: 3px 8px;
      border-radius: 4px;
      font-size: 12px;
      margin-left: 5px;
      margin-bottom: 5px;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="header">
      <img src="{{baseUrl}}/images/logo.png" alt="Logo" class="logo">
      <h1>{{report.title}}</h1>
      <div class="date">                    : {{formatDate report.createdAt}}</div>
    </div>

    <div class="section">
      <h2 class="section-title">                    </h2>
```

```
    <table class="info-table">
      <tr>
        <th>          </th>
        <td>{{report.location}}</td>
      </tr>
      <tr>
        <th>                  </th>
        <td>{{formatDate report.inspectionDate}}</td>
      </tr>
      <tr>
        <th>                      </th>
        <td>
          <span class="severity severity-{{report.severity}}">
            {{#ifEquals report.severity "high"}}         {{/ifEquals}}
            {{#ifEquals report.severity "medium"}}           {{/ifEquals}}
            {{#ifEquals report.severity "low"}}          {{/ifEquals}}
          </span>
        </td>
      </tr>
      <tr>
        <th>                                  </th>
        <td>{{report.analysisResults.total_defects}}</td>
      </tr>
    </table>

    {{#if report.description}}
      <h2 class="section-title">                 </h2>
      <p>{{report.description}}</p>
    {{/if}}

    {{#if report.tags}}
      <div class="tags">
        {{#each report.tags}}
          <span class="tag">{{this}}</span>
        {{/each}}
      </div>
    {{/if}}
  </div>

  {{#if report.analysisResults.image}}
    <div class="section">
      <h2 class="section-title">                      </h2>
      <div class="image-container">
        <img src="{{report.analysisResults.image}}" alt="Analyzed Image"
class="analyzed-image">
      </div>
    </div>
  {{/if}}

  <div class="section">
    <h2 class="section-title">                 </h2>
```

```
    {{#if report.analysisResults.defect_summary}}
      <div class="defect-summary">
        {{#if report.analysisResults.defect_summary.crack}}
          <div class="defect-card crack">
            <div class="defect-count">{{report.analysisResults.defect_summary.crack}}
</div>
            <div class="defect-type">          </div>
          </div>
        {{/if}}

        {{#if report.analysisResults.defect_summary.corrosion}}
          <div class="defect-card corrosion">
            <div class="defect-
count">{{report.analysisResults.defect_summary.corrosion}}</div>
            <div class="defect-type">       </div>
          </div>
        {{/if}}

        {{#if report.analysisResults.defect_summary.exposed_rebar}}
          <div class="defect-card exposed_rebar">
            <div class="defect-
count">{{report.analysisResults.defect_summary.exposed_rebar}}</div>
            <div class="defect-type">                          </div>
          </div>
        {{/if}}
      </div>
    {{else}}
      <p>                              .</p>
    {{/if}}
  </div>

  {{#if report.analysisResults.defects}}
    <div class="section">
      <h2 class="section-title">                    </h2>
      <table class="defects-table">
        <thead>
          <tr>
            <th>#</th>
            <th>              </th>
            <th>                </th>
            <th>           </th>
            <th>           </th>
          </tr>
        </thead>
        <tbody>
          {{#each report.analysisResults.defects}}
            <tr>
              <td>{{@index}}</td>
              <td>
                {{#ifEquals this.type "crack"}}       {{/ifEquals}}
                {{#ifEquals this.type "corrosion"}}       {{/ifEquals}}
                {{#ifEquals this.type "exposed_rebar"}}                          {{/ifEquals}}
```

```
            </td>
            <td>{{multiply this.confidence 100}}%</td>
            <td>
              {{#if this.dimensions}}
                {{round this.dimensions.width}} × {{round this.dimensions.height}}
              {{else}}

              {{/if}}
            </td>
            <td>
              <span class="severity
                {{#if (gt this.confidence 0.8)}}severity-high{{/if}}
                {{#if (and (gt this.confidence 0.5) (lte this.confidence 0.8))}}severity-
medium{{/if}}
                {{#if (lte this.confidence 0.5)}}severity-low{{/if}}
              ">
                {{#if (gt this.confidence 0.8)}}        {{/if}}
                {{#if (and (gt this.confidence 0.5) (lte this.confidence 0.8))}}          {{/if}}
                {{#if (lte this.confidence 0.5)}}          {{/if}}
              </span>
            </td>
          </tr>
        {{/each}}
      </tbody>
    </table>
  </div>
 {{/if}}

 <div class="section">
  <h2 class="section-title">            </h2>
  <div class="recommendations">
   {{#if report.recommendations}}
    {{report.recommendations}}
   {{else}}
                              .
   {{/if}}
  </div>
 </div>

 <div class="footer">
  <p>
        </p>
  <p>                  : {{formatDate report.createdAt}}</p>
 </div>
</div>
</body>
</html>
```

4.

4.1 (reportService.js)

```
// frontend/src/services/reportService.js
import axios from 'axios';

const API_URL = process.env.REACT_APP_API_URL || 'http://localhost:8000';

/**
 *
 * @param {Object} reportData -
 * @returns {Promise<Object>} -
 */
export const createReport = async (reportData) => {
  try {
    const response = await axios.post(`${API_URL}/api/reports`, reportData, {
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${localStorage.getItem('token')}`
      }
    });

    return response.data;
  } catch (error) {
    console.error('Error creating report:', error);
    throw error;
  }
};

/**
 *
 * @param {string} reportId -
 * @returns {Promise<Object>} -
 */
export const getReport = async (reportId) => {
  try {
    const response = await axios.get(`${API_URL}/api/reports/${reportId}`, {
      headers: {
        'Authorization': `Bearer ${localStorage.getItem('token')}`
      }
    });

    return response.data;
  } catch (error) {
    console.error('Error getting report:', error);
    throw error;
  }
};
```

```
/**
 *
 * @param {Object} filters -
 * @returns {Promise<Object>} -
 */
export const getReports = async (filters = {}) => {
  try {
    //
    const queryParams = new URLSearchParams();

    if (filters.severity) {
      queryParams.append('severity', filters.severity);
    }

    if (filters.projectId) {
      queryParams.append('projectId', filters.projectId);
    }

    if (filters.startDate) {
      queryParams.append('startDate', filters.startDate);
    }

    if (filters.endDate) {
      queryParams.append('endDate', filters.endDate);
    }

    if (filters.limit) {
      queryParams.append('limit', filters.limit);
    }

    const queryString = queryParams.toString();
    const url = queryString ? `${API_URL}/api/reports?${queryString}` : `${API_URL}/
api/reports`;

    const response = await axios.get(url, {
      headers: {
        'Authorization': `Bearer ${localStorage.getItem('token')}`
      }
    });

    return response.data;
  } catch (error) {
    console.error('Error getting reports:', error);
    throw error;
  }
};

/**
 *
 * @param {string} reportId -
 * @param {Object} updateData -
 * @returns {Promise<Object>} -
```

```
 */
export const updateReport = async (reportId, updateData) => {
  try {
    const response = await axios.put(`${API_URL}/api/reports/${reportId}`,
updateData, {
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${localStorage.getItem('token')}`
      }
    });

    return response.data;
  } catch (error) {
    console.error('Error updating report:', error);
    throw error;
  }
};

/**
 *
 * @param {string} reportId -
 * @returns {Promise<Object>} -
 */
export const deleteReport = async (reportId) => {
  try {
    const response = await axios.delete(`${API_URL}/api/reports/${reportId}`, {
      headers: {
        'Authorization': `Bearer ${localStorage.getItem('token')}`
      }
    });

    return response.data;
  } catch (error) {
    console.error('Error deleting report:', error);
    throw error;
  }
};

/**
 *                        PDF
 * @param {string} reportId -
 */
export const downloadReport = async (reportId) => {
  try {
    const response = await axios.get(`${API_URL}/api/reports/${reportId}/
download`, {
      headers: {
        'Authorization': `Bearer ${localStorage.getItem('token')}`
      },
      responseType: 'blob'
    });
```

```
    //
    const url = window.URL.createObjectURL(new Blob([response.data]));
    const link = document.createElement('a');
    link.href = url;

    //
    const contentDisposition = response.headers['content-disposition'];
    let filename = 'report.pdf';

    if (contentDisposition) {
      const filenameMatch = contentDisposition.match(/filename="(.+)"/);
      if (filenameMatch.length === 2) {
        filename = filenameMatch[1];
      }
    }

    link.setAttribute('download', filename);
    document.body.appendChild(link);
    link.click();

    //
    link.remove();
    window.URL.revokeObjectURL(url);
  } catch (error) {
    console.error('Error downloading report:', error);
    throw error;
  }
};

/**
 *
 * @param {string} reportId -
 * @param {Array} recipients -
 * @returns {Promise<Object>} -
 */
export const shareReport = async (reportId, recipients) => {
  try {
    const response = await axios.post(`${API_URL}/api/reports/${reportId}/share`, {
recipients }, {
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${localStorage.getItem('token')}`
      }
    });

    return response.data;
  } catch (error) {
    console.error('Error sharing report:', error);
    throw error;
  }
};
```

5.

5.1 (ar/translation.json)

```
{
 "reports": {
  "title": "          ",
  "createNew": "                    ",
  "tabs": {
   "all": "                ",
   "recent": "                    ",
   "shared": "                  "
  },
  "createSuccess": "                          ",
  "createError": "                            ",
  "defaultTitle": "                  - {date}",
  "generator": {
   "title": "                    ",
   "autoFill": "                  ",
   "title": "                ",
   "location": "          ",
   "inspectionDate": "              ",
   "description": "                ",
   "severity": "                  ",
   "recommendations": "            ",
   "tags": "          ",
   "addTag": "              ",
   "add": "        ",
   "creating": "                ...",
   "create": "                  "
  },
  "list": {
   "title": "                  ",
   "search": "                    ",
   "filter": "        ",
   "location": "          ",
   "date": "          ",
   "noReports": "                        ",
   "noSearchResults": "                                ",
   "fetchError": "                            ",
   "deleteError": "                            ",
   "view": "      ",
   "download": "        ",
   "share": "          ",
   "delete": "      "
  },
  "viewer": {
   "print": "        ",
   "share": "          ",
   "download": "        ",
```

```
      "downloading": "                    ...",
      "close": "        "
    },
    "template": {
      "details": "                      ",
      "location": "          ",
      "inspectionDate": "                  ",
      "severity": "                      ",
      "totalDefects": "                      ",
      "description": "                    ",
      "analyzedImage": "                        ",
      "defectSummary": "                    ",
      "noDefectsFound": "                                  ",
      "detailedDefects": "                      ",
      "defectType": "                ",
      "confidence": "                  ",
      "dimensions": "             ",
      "severity": "             ",
      "recommendations": "               ",
      "noRecommendations": "                                ",
      "createdOn": "                       ",
      "notSpecified": "              ",
      "notAvailable": "              ",
      "footer": "
          - {date}"
    },
    "recommendations": {
      "noDefects":
"                                          6                                        .                                                   ,"
      "highSeverity": "                                .
                                                 .,"
      "mediumSeverity": "                    .
                                                                 ,"
      "lowSeverity": "                                    .
        6                                        .,"
      "cracks": "                                                                                                  ,"
      "corrosion": "
           .",
      "exposedRebar": "
                   ."
    }
  }
}
```

5.2 (en/translation.json)

```
{
  "reports": {
    "title": "Reports",
    "createNew": "Create New Report",
```

```
  "tabs": {
    "all": "All Reports",
    "recent": "Recent Reports",
    "shared": "Shared Reports"
  },
  "createSuccess": "Report created successfully",
  "createError": "Error creating report",
  "defaultTitle": "Structural Inspection Report - {date}",
  "generator": {
    "title": "Create New Report",
    "autoFill": "Auto Fill",
    "title": "Report Title",
    "location": "Location",
    "inspectionDate": "Inspection Date",
    "description": "Report Description",
    "severity": "Severity Level",
    "recommendations": "Recommendations",
    "tags": "Tags",
    "addTag": "Add Tag",
    "add": "Add",
    "creating": "Creating...",
    "create": "Create Report"
  },
  "list": {
    "title": "Reports List",
    "search": "Search reports",
    "filter": "Filter",
    "location": "Location",
    "date": "Date",
    "noReports": "No reports available",
    "noSearchResults": "No results match your search",
    "fetchError": "Error fetching reports",
    "deleteError": "Error deleting report",
    "view": "View",
    "download": "Download",
    "share": "Share",
    "delete": "Delete"
  },
  "viewer": {
    "print": "Print",
    "share": "Share",
    "download": "Download",
    "downloading": "Downloading...",
    "close": "Close"
  },
  "template": {
    "details": "Report Details",
    "location": "Location",
    "inspectionDate": "Inspection Date",
    "severity": "Severity Level",
    "totalDefects": "Total Defects",
    "description": "Report Description",
```

```
    "analyzedImage": "Analyzed Image",
    "defectSummary": "Defect Summary",
    "noDefectsFound": "No defects found",
    "detailedDefects": "Detailed Defects",
    "defectType": "Defect Type",
    "confidence": "Confidence",
    "dimensions": "Dimensions",
    "severity": "Severity",
    "recommendations": "Recommendations",
    "noRecommendations": "No specific recommendations",
    "createdOn": "Created On",
    "notSpecified": "Not specified",
    "notAvailable": "Not available",
    "footer": "This report was generated by the AI Structural Defect Detection
System - {date}"
  },
  "recommendations": {
    "noDefects": "No defects were detected in the structure. We recommend
conducting a periodic inspection every 6 months to ensure the structure's
integrity.",
    "highSeverity": "High severity defects were detected in the structure. We
recommend immediate repairs to prevent further deterioration and ensure
structural safety.",
    "mediumSeverity": "Medium severity defects were detected in the structure.
We recommend scheduling repairs within the next three months and monitoring
the situation regularly.",
    "lowSeverity": "Low severity defects were detected in the structure. We
recommend monitoring these defects regularly and conducting a follow-up
inspection after 6 months.",
    "cracks": "Treat cracks using appropriate injection materials and approved
repair techniques.",
    "corrosion": "Remove rust and corrosion and apply a suitable protective layer
to prevent further deterioration.",
    "exposedRebar": "Cover exposed reinforcement bars with adequate concrete
layer and apply corrosion-resistant materials."
  }
 }
}
```

6.

```
 -- database/schema/reports.sql

--
CREATE TABLE reports (
  id SERIAL PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  location VARCHAR(255),
```

```
  inspection_date DATE DEFAULT CURRENT_DATE,
  severity VARCHAR(50) DEFAULT 'medium',
  recommendations TEXT,
  tags JSONB DEFAULT '[]',
  analysis_results JSONB,
  project_id INTEGER REFERENCES projects(id) ON DELETE SET NULL,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  pdf_url VARCHAR(255),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

--
CREATE INDEX idx_reports_user_id ON reports(user_id);
CREATE INDEX idx_reports_project_id ON reports(project_id);
CREATE INDEX idx_reports_created_at ON reports(created_at);
CREATE INDEX idx_reports_severity ON reports(severity);

--                      updated_at
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
  NEW.updated_at = CURRENT_TIMESTAMP;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

--                      updated_at
CREATE TRIGGER update_reports_updated_at
BEFORE UPDATE ON reports
FOR EACH ROW
EXECUTE FUNCTION update_updated_at_column();

--
CREATE TABLE report_shares (
  id SERIAL PRIMARY KEY,
  report_id INTEGER NOT NULL REFERENCES reports(id) ON DELETE CASCADE,
  shared_by INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  shared_with INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  permissions VARCHAR(50) DEFAULT 'read', -- read, edit, etc.
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

--
CREATE INDEX idx_report_shares_report_id ON report_shares(report_id);
CREATE INDEX idx_report_shares_shared_with ON report_shares(shared_with);
```

7.

. :

1. .
2. .
3. .
4. **PDF** PDF .
5. **RESTful** .
6. .
7. .

.